S-PLUS 6.1 for UNIX/Linux Supplement

May 2002

Insightful Corporation
Seattle, Washington

Proprietary Notice

Insightful Corporation owns both this software program and its documentation. Both the program and documentation are copyrighted with all rights reserved by Insightful Corporation.

The correct bibliographical reference for this document is as follows:

S-PLUS 6.1 for UNIX Supplement, Insightful Corporation, Seattle, WA.

Printed in the United States.

Copyright Notice

Copyright © 1987-2002, Insightful Corporation. All rights reserved.

Insightful Corporation
1700 Westlake Avenue N, Suite 500
Seattle, WA 98109-3044
USA

ACKNOWLEDGMENTS

S-PLUS would not exist without the pioneering research of the Bell Labs S team at AT&T (now Lucent Technologies): John Chambers, Richard A. Becker (now at AT&T Laboratories), Allan R. Wilks (now at AT&T Laboratories), Duncan Temple Lang, and their colleagues in the statistics research departments at Lucent: William S. Cleveland, Trevor Hastie (now at Stanford University), Linda Clark, Anne Freeny, Eric Grosse, David James, José Pinheiro, Daryl Pregibon, and Ming Shyu.

Insightful Corporation thanks the following individuals for their contributions to this and earlier releases of S-PLUS: Douglas M. Bates, Leo Breiman, Dan Carr, Steve Dubnoff, Don Edwards, Jerome Friedman, Kevin Goodman, Perry Haaland, David Hardesty, Frank Harrell, Richard Heiberger, Mia Hubert, Richard Jones, Jennifer Lasecki, W.Q. Meeker, Adrian Raftery, Brian Ripley, Peter Rousseeuw, J.D. Spurrier, Anja Struyf, Terry Therneau, Rob Tibshirani, Katrien Van Driessen, William Venables, and Judy Zeh.

CONTENTS

Acknowledgments iii

Chapter 1 Welcome 1

- **Welcome to S-PLUS 6.1!** 2
- **Installation** 3
- **What's New in S-PLUS 6.1** 8
- **Help, Support, and Learning Resources** 10
- **Typographic Conventions** 18

Chapter 2 CONNECT/C++ 19

- **Introduction** 20
- **Simple Examples: An Application and a Called Routine** 21
- **CONNECT/C++ Class Overview** 28
- **CONNECT/C++ Architectural Features** 31

Chapter 3 Globalization 43

- **Introduction** 44
- **Working With Locales** 45
- **Using Extended Characters** 48
- **Importing, Exporting, and Displaying Numeric Data** 50

Chapter 4 User-Defined Help **53**

Introduction 54

Creating, Editing, and Distributing a Help File 56

Common Text Formats 61

Contents of Help Files 63

Chapter 5 Oracle Connectivity **81**

Introduction 82

Reading Data From an Oracle Database 83

Writing Data to an Oracle Database 90

Executing Arbitrary SQL Statements 92

Index **95**

WELCOME 1

Welcome to S-PLUS 6.1! **2**

Installation **3**
- Supported Platforms and System Requirements 3
- Installation Instructions 4
- Running S-PLUS 5

What's New in S-PLUS 6.1 **8**
- CONNECT/C++ 8
- Globalization 8
- User-Defined Help 9
- Oracle Connectivity (Solaris Only) 9

Help, Support, and Learning Resources **10**
- Online Help 10
- Online Manuals 13
- S-PLUS on the Web 14
- Training Courses 14
- Technical Support 14
- Books Using S-PLUS 15

Typographic Conventions **18**

WELCOME TO S-PLUS 6.1!

S-PLUS 6.1 is a service upgrade for S-PLUS 6, a significant new release of S-PLUS based on the latest version of the powerful, object-oriented S language developed at Lucent Technologies. S is a rich environment designed for interactive data discovery and is the only language created specifically for data visualization and exploration, statistical modeling, and programming with data.

S-PLUS 6 continues to be the premier solution for your data analysis and technical graphing needs. The Java graphical user interface gives you point-and-click access to graphing and statisticsWith S-PLUS 6, you can also program interactively using the S-PLUS programming language.

In a typical S-PLUS session, you can:

- Import data from virtually any source.
- View your data in a convenient **Data** window.
- Create plots with the click of a button.
- Control details of your graphics to produce stunning, professional-looking output for export to your report documents.
- Perform statistical analyses from convenient dialogs in the menu system.
- Create your own functions.
- Completely customize your user interface.

INSTALLATION

Supported Platforms and System Requirements

S-PLUS 6.1 for UNIX/Linux is supported on the following:

- 32-bit Solaris, Solaris 2.6, 7, or 8
- Red Hat Linux 6.1, 6.2, 7.0, 7.2

Before installing S-PLUS, review the minimum system configuration information in Table 1.1. To determine the required RAM, sum the base RAM and the per-user RAM multiplied by the number of simultaneous users. For example, a single-user Linux system should have at least 64 + 32*1 = 96MB of RAM.

Table 1.1: *Minimum system configuration information for S-PLUS 6.1.*

Platform	Operating System	Disk Space	Base RAM	Per-User RAM
Sun SPARC	Solaris 2.6, 7, 8	225MB	64MB	40MB
Intel x86	Red Hat 6.1, 6.2, 7.0, 7.2	225MB	64MB	32MB

The base RAM and per-user RAM listings can also be used to calculate minimum swap space requirements. In general, the minimum swap space required is twice the sum of the base RAM and the per-user RAM multiplied by the number of simultaneous users. For example, on a Solaris system with three simultaneous S-PLUS users, the minimum swap space is 2*(64+(40*3)) = 368MB.

These values are minima; if you work with moderately-sized data sets, the numbers may be insufficient for your needs. Note also that they are computed for simple command-line S-PLUS. If you intend to run the Java GUI, you should add approximately 100MB RAM per user to the numbers shown in Table 1.1. For example, on Solaris, Insightful Corporation recommends approximately 200MB RAM for one user running the Java GUI.

Linux Platforms

S-PLUS 6.1 was developed under Red Hat 6.1 and tested/supported on Red Hat 6.2, 7.0, and 7.2. These releases contain the kernels and compilers listed in Table 1.2. Other Linux distributions, such as SuSE, are not officially supported or tested. However, if you want to use another Linux distribution, you should use one that has similar

versions of the kernels and compilers (for example, SuSE 6.4, 7.0, or 7.2). For more information on various Linux distributions, see **http://www.distrowatch.com**.

Table 1.2: *Kernels and compilers for the supported versions of Red Hat Linux.*

Red Hat OS	Kernel	GCC	glibc
7.2	2.4.7	2.96	2.2.4
7.0	2.2.16	2.96	2.1.92
6.2	2.2.14	2.91.66	2.1.3
6.1	2.2.12	2.91.66	2.1.2

Java Runtime Environment (JRE)

The Java runtime environment (JRE) version 1.3 is included in S-PLUS 6.1. Your operating system must support JRE 1.3 to run the Java-enabled version of S-PLUS. The JRE provided by S-PLUS is installed as part of the S-PLUS distribution, and under normal circumstances it is used only by S-PLUS. If you have a different version of the JRE on your system, the JRE used by S-PLUS should not interfere with your other JRE applications, which will continue to use the version you've previously installed.

See the S-PLUS release notes for specific information regarding the JRE on your platform. In particular, Solaris operating environments require various patches from Sun to run Java 1.3. The S-PLUS 6.1 release notes contain pointers to Web site where you can download these patches.

Installation Instructions

For complete installation instructions, please see the plain text file **INSTALL.TXT** located at the top level of your S-PLUS 6.1 CD-ROM, or the manual *S-PLUS 6 Installation and Maintenance Guide*.

During installation, you are prompted for your S-PLUS serial number. Note that the serial number format has changed dramatically in S-PLUS 6.1; you can find your new serial number on the S-PLUS 6.1 CD-ROM jewel case. The serial number you enter during installation is recorded and you can retrieve it later by typing `Splus SERIAL` at the UNIX prompt on your system.

Do not install this release over any existing version of S-PLUS. Instead, designate a clean installation directory for S-PLUS 6.1 and proceed with the installation as described in either **INSTALL.TXT** or the manual.

Running S-PLUS

Before starting S-PLUS, you must do the following:

1. Set your **DISPLAY** environment variable to your local machine.
2. Create an S-PLUS chapter to hold your work.

Setting your **DISPLAY** environment variable is necessary for the Java features in S-PLUS. To set your display from a C-like shell (**csh**, **tcsh**, etc.), use the **setenv** command from the UNIX prompt:

```
setenv DISPLAY <display_name>
```

where *<display_name>* is the name of your local machine. From the Bourne- and Korn-like shells (including **sh**, **ksh**, **bash**, etc.), use the following commands:

```
DISPLAY=<display_name>;export DISPLAY
```

Creating an S-PLUS chapter is necessary for storing the data objects and external files you create in S-PLUS. The following commands create an S-PLUS chapter named **mysplus** for you to work in (be sure you don't have a **mysplus** directory in your home directory before typing these commands).

```
cd
mkdir mysplus
cd mysplus
Splus CHAPTER
```

You are now ready to start S-PLUS. S-PLUS may be launched in a variety of modes. The following lists each mode and the corresponding UNIX command-line expression for launching it. In all of the commands below, **Splus** refers to the script you use to launch S-PLUS 6.1 on your system.

- S-PLUS command line without Java

  ```
  Splus
  ```

- S-PLUS command line supporting Java calls, Java graphics, and the Java help interface

```
Splus -j
```

- S-PLUS graphical user interface

```
Splus -g
```

or

```
Splus -g &
```

The second command using the ampersand (`&`) puts the GUI process into the background, freeing your xterm for other uses.

The **-e** flag may be added to either of the first two modes to enable command-line editing. The **Commands** window in the graphical user interface always allows basic editing.

Note

Java has a number of known problems displaying on a remote display using the X Window System. For this reason, Insightful Corporation suggests (if possible) that you limit your use of the Java-based GUI in this release to the console of the machine on which S-PLUS is installed; that is, treat this release of the Java-based GUI as a workstation release, rather than as a network client/server release.

For specific details describing the limitations of Java GUI clients on remote displays, see the S-PLUS 6.1 release notes.

S-PLUS 6 includes two additional flags, **-jit** and **-helpoff**:

- The **-jit** flag works with the **-g**, **-j**, and **-userapp** flags and allows you to turn on the Java just-in-time compiler. This makes the graphical user interface and help system run faster but introduces instabilities that often lead to crashes. In particular, the just-in-time compiler often crashes while repainting graphical user interface elements such as the JavaHelp window and the **Data** window.
- The **-helpoff** flag is useful only with the **-g** flag. It turns off the automatic invisible startup of the help system. The invisible startup improves initial responsiveness of the help system but

adds a significant memory footprint to the current session. If you want to optimize your available memory, this flag may prove useful.

WHAT'S NEW IN S-PLUS 6.1

In this section, we briefly describe the new features in S-PLUS 6.1. Detailed information can be found in the relevant chapters of this supplement. For information on other enhancements and improvements in S-PLUS 6.1, see the release notes.

CONNECT/C++

CONNECT/C++ is now available with S-PLUS for UNIX/Linux. Previously, this feature was available only in S-PLUS for Windows. You can use CONNECT/C++ to create C++ applications that access S-PLUS functionality as well as C++ functions that are called from S-PLUS via the `.Call` interface. S-PLUS 6.1 includes a C++ class library you can use to create and manipulate S-PLUS functions, parse and evaluate S-PLUS expressions, and return messages when objects have changed or databases have been attached or detached.

The CONNECT/C++ library is located in **$SHOME/newfun/lib**, source code can be found in **$SHOME/sconnect**, and examples are in **$SHOME/sconnect/samples**. Here, **$SHOME** is the home directory for your installation of S-PLUS 6.1. You can see this directory path by typing either `getenv("SHOME")` at the command prompt in S-PLUS or `Splus SHOME` from the UNIX prompt, where **Splus** is the script you use to launch S-PLUS 6.1.

Globalization

Aspects of S-PLUS have been significantly enhanced to support globalization and locales. The concept of a *locale* is akin to **Regional Options** (or **Regional Settings**) in Microsoft Windows; setting a locale is a way of specifying which character sets, number formats, and collation sequences to use. This allows S-PLUS to support Western European character sets (ISO 8859-1) as well as 8-bit ASCII characters (ASCII codes 128-255) in character data and the names of objects. Globalization also permits you to import, export, and display numbers written with decimal markers other than the period, and thousands separators other than the comma. S-PLUS 6.1 fully supports French and German locales.

Note

The implementation of locales within S-PLUS does not apply to date formats because they are handled through a separate mechanism in the `timeDate` set of functions. Also, it does not apply to messaging; all messages in S-PLUS are in the English language.

User-Defined Help

In S-PLUS 6.1, the process for creating and distributing custom S-PLUS help files has been greatly streamlined and better documented.

Oracle Connectivity (Solaris Only)

S-PLUS 6.1 expands connectivity with Oracle database clients. This allows you to read data from an Oracle table, insert data into a table, and run arbitrary SQL queries, all from the S-PLUS command line.

This feature is available on Solaris installations of S-PLUS 6.1 only; Linux installations do not support this feature.

HELP, SUPPORT, AND LEARNING RESOURCES

There are a variety of ways to accelerate your progress with S-PLUS. This section describes the learning and support resources available to S-PLUS users.

Online Help

S-PLUS offers an online JavaHelp system to make learning and using S-PLUS easier. Under the **Help** menu in the S-PLUS GUI, you will find detailed help on each function in the S-PLUS language. You can access the help system from the S-PLUS prompt or the **Commands** window in the GUI by typing `help.start()`.

Context-sensitive help is available by clicking the **Help** buttons in dialogs.

JavaHelp

JavaHelp in S-PLUS uses Java to display the help files. To access JavaHelp, do one of the following:

- From the main menu in the S-PLUS GUI, choose **Help ► Contents**, **Help ► Index**, or **Help ► Search** to view the help system's table of contents, index, and search pages, respectively.
- From the S-PLUS prompt or the **Commands** window in the GUI, type `help.start()`.

To turn the help system off, type `help.off()` at the S-PLUS prompt.

As shown in Figure 1.1, the JavaHelp window has three main areas: the *toolbar*, the *navigation pane*, and the *topic pane*.

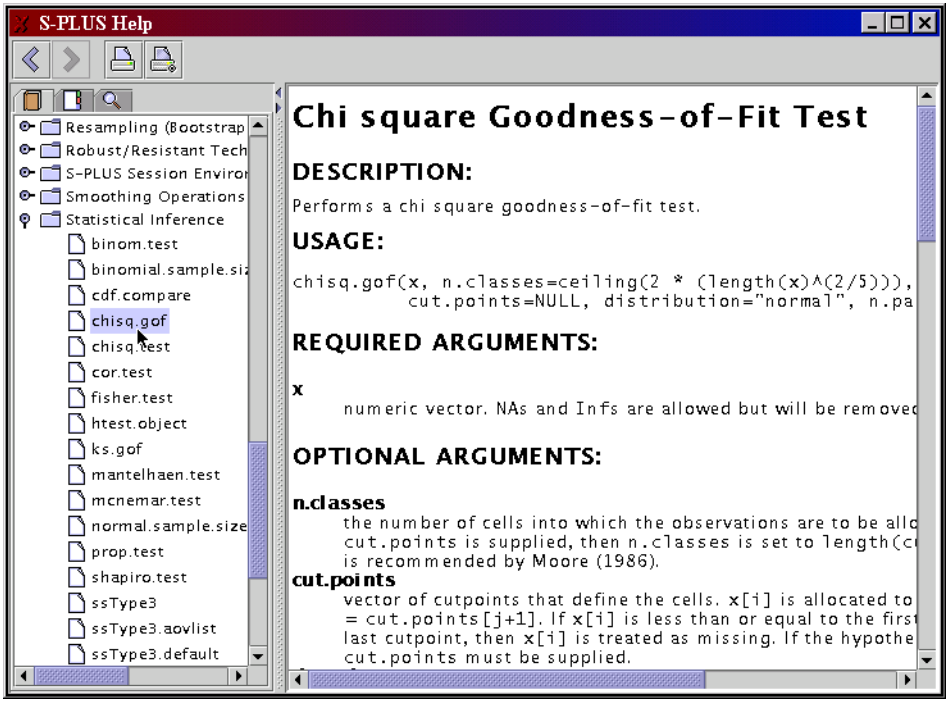

Figure 1.1: *The S-PLUS JavaHelp window.*

Using the toolbar

Table 1.3 lists the four buttons on the help window toolbar.

Table 1.3: *Toolbar buttons in the JavaHelp window.*

Button	Description
Previous	Returns to previously viewed help topic.
Next	Moves to next help topic in a previously-displayed sequence of topics.
Print	Prints the current help topic.

Table 1.3: *Toolbar buttons in the JavaHelp window. (Continued)*

Button	Description
Page Setup	Determines the orientation of the page for printing purposes.

Using the navigation pane

The navigation pane appears on the left side of the JavaHelp window. Like the help window itself, the left pane is divided into three parts: the **Table of Contents** , **Index** , and **Search** pages:

- The **Table of Contents** page organizes help topics by category so related help files can be found easily. These categories appear as small folder icons, labeled with the name of the category. To open a category, double-click the icon or label. To select a topic within the category, double-click its page icon or the topic title.
- The **Index** page lists available help topics by keyword. Keywords are typically function names for S-PLUS language functions. Type a word in the text box and press ENTER to find the keywords that most closely match it.
- The **Search** tab provides a full-text search for the entire help system. Type the word or phrase you want to find in the text box and press ENTER. JavaHelp displays in the list box all help files containing that keyword. Double-click a title to display the desired help topic.

Using the topic pane

The topic pane appears on the right side of the help window and displays the help topics you choose. It usually appears with both vertical and horizontal scroll bars, but you can expand the JavaHelp window to increase the width of the right pane. Many help files are too long to be fully displayed in a single screen, so choose a convenient height for your JavaHelp window and then use the vertical scroll bars to scroll through the text.

Help at the Command Line

When working from the S-PLUS command line, you can obtain help for any S-PLUS function using the `help` or `?` functions. For example, to open the help file for `anova`, simply type:

```
> help(anova)
```

or

```
> ?anova
```

Online Manuals

In addition to this supplement, the booklet *Getting Started with S-PLUS 6 for UNIX/Linux*, the *User's Guide*, the *Programmer's Guide*, and both volumes of the *Guide to Statistics* are available online. *Getting Started with S-PLUS 6* provides a tutorial introduction to the product and so is particularly useful for those new to S-PLUS.

To view a manual online, navigate to **$SHOME/doc** and open the desired PDF file. See Table 1.4 for the names of the PDF files.

Table 1.4: *Online documentation available in **$SHOME/doc**.*

Manual	PDF File
S-PLUS 6.1 for UNIX/Linux Supplement	**unixsupp61.pdf**
S-PLUS 6 Installation and Maintenance Guide	**instman.pdf**
Getting Started with S-PLUS 6 for UNIX/Linux	**getstart.pdf**
S-PLUS 6 User's Guide	**unixug.pdf**
S-PLUS 6 Guide to Statistics, Volume 1	**statman1.pdf**
S-PLUS 6 Guide to Statistics, Volume 2	**statman2.pdf**
S-PLUS 6 Programmer's Guide	**pguide.pdf**

Note
The online manuals are viewed using Adobe Acrobat Reader. It is generally useful to turn on *bookmarks* under the **View** entry of the menu bar while using Acrobat Reader, rather than rely on the contents at the start of the manuals. Bookmarks are always visible and can be expanded and collapsed to show just chapter titles or to include section headings.

S-PLUS on the Web

You can find S-PLUS on the World Wide Web at **http://www.insightful.com/support**. In these pages, you will find a variety of information, including:

- Product information.
- FAQ pages.
- The most recent service packs.
- Training course information.
- Information on classroom use and related educational materials.

Training Courses

Insightful Corporation's Educational Services offers a number of courses designed to quickly make you efficient and effective at analyzing data with S-PLUS. The courses are taught by professional statisticians and leaders in statistical fields. Courses feature a hands-on approach to learning, dividing class time between lecture and online exercises. All participants receive the educational materials used in the course, including lecture notes, supplementary materials, and exercise data. For further information, see

http://www.insightful.com/services/training.asp.

Technical Support

North, Central, and South America

Contact Technical Support at Insightful Corporation:

Telephone: 206.283.8802 or 1.800.569.0123, ext. 235, Monday-Friday, 6:00 a.m. PST (9:00 a.m. EST) to 5:00 p.m. PST (8:00 p.m. EST)

Fax: 206.283.8691

E-mail: support@insightful.com

Web: **http://www.insightful.com/support**

All Other Locations

Contact the European Headquarters of Insightful Corporation:

Christoph Merian-Ring 11, 4153 Reinach, Switzerland

Telephone: +41 61 717 9340

Fax: +41 61 717 9341

E-mail: info@ch.insightful.com

Books Using S-PLUS

Introductory

Burns, P. (1998). *S Poetry*. Download for free from

http://www.seanet.com/~pburns/Spoetry

Krause, A. and Olson, M. (2000). *The Basics of S and S-PLUS* (2nd ed.). Springer-Verlag, New York.

Spector, P. (1994). *An Introduction to S and S-PLUS*. Duxbury Press, Belmont, CA.

Statistics and Data Analysis (General)

Alzola, C.F. and Harrell, F.E. (2001). *An Introduction to S-PLUS and the Hmisc and Design Libraries*. Download for free from

http://hesweb1.med.virginia.edu/biostat/s/doc/splus.pdf

Atkinson, A., and Riani, M. (2000). *Robust Diagnostic Regression Analysis*. Springer-Verlag.

Bowman, A. and Azzalini, A. (1997). *Applied Smoothing Techniques for Data Analysis: The Kernel Approach with S-PLUS Illustrations*. Clarendon Press.

Chambers, J.M. and Hastie, T.J. (1992). *Statistical Models in S*. Chapman and Hall.

Crawley, M. (2002). *Statistical Computing: An Introduction to Data Analysis Using S-PLUS*. Wiley.

Davison, A.C. and Hinkley, D.V. (1997). *Bootstrap Methods and Their Applications*. Cambridge University Press.

Efron, B. and Tibshirani, R.J. (1994). *An Introduction to the Bootstrap*. Chapman & Hall.

Everitt, B. (1994). *A Handbook of Statistical Analyses Using S-PLUS*. Chapman & Hall, London.

Härdle, W. (1991). *Smoothing Techniques with Implementation in S*. Springer-Verlag, New York.

Hastie, T.J. and Tibshirani, R.J. (1990). *Generalized Additive Models*. Chapman & Hall.

Hastie, T.J., Tibshirani, R.J., and Friedman, J. (2001). *The Elements of Statistical Learning: Data Mining, Inference and Prediction*. Springer-Verlag, New York.

Huet, S., et al. (1996). *Statistical Tools for Nonlinear Regression: A Practical Guide with S-PLUS Examples*. Springer-Verlag, New York.

Marazzi, A. (1992). *Algorithms, Routines and S Functions for Robust Statistics*. Wadsworth & Brooks/Cole, Pacific Grove, CA.

Newton, H.J. and Harvill, J.L. (1997). *StatConcepts: A Visual Tour of Statistical Ideas*. Duxbury Press.

Nolan, D. and Speed, T.P. (2000). *Stat Labs: Mathematical Statistics Through Applications*. Springer-Verlag.

Pinheiro, J.C. and Bates, D.M. (2000). *Mixed-Effects Models in S and S-PLUS*. Springer-Verlag, New York.

Schumaker, R.E. and Akers, A. (2001). *Understanding Statistical Concepts Using S-PLUS*. Lawrence Erlbaum Associates, Mahwah, NJ.

Venables, W.N. and Ripley, B.D. (1999). *Modern Applied Statistics with S-PLUS* (3rd ed.). Springer-Verlag, New York.

Wilcox, R. (1997). *Introduction to Robust Estimation and Hypothesis Testing*. Academic Press.

Statistics and Data Analysis (Specialized)

Bruce, A. and Gao, H.-Y. (1996). *Applied Wavelet Analysis with S-PLUS*. Springer-Verlag, New York.

Everitt, B. and Rabe-Hesketh, S. (2001). *Analyzing Medical Data Using S-PLUS (Statistics for Biology and Health)*. Springer-Verlag.

Harrell, F.E. (2001). *Regression Modeling Strategies with Applications to Linear Models, Logistic Regression, and Survival Analysis*. Springer-Verlag.

Kaluzny, S.P., Vega, S.C., Cardoso, T.P., and Shelly, A.A. (1997). *S+SpatialStats User's Manual*. Springer-Verlag, New York.

Meeker, W. and Escobar, L. (1998). *Statistical Methods for Reliability Data*. John Wiley & Sons, Inc.

Millard, S.P. and Krause, A. (2001). *Applied Statistics in the Pharmaceutical Industry*. Springer-Verlag, New York.

Millard, S.P. and Neerchal, N.K. (2000). *Environmental Statistics with S-PLUS*. CRC Press.

Selvin, S. (1998). *Modern Applied Biostatistical Methods: Using S-PLUS*. Oxford University Press.

Therneau, T.M. and Grambsch, P.M. (2000). *Modeling Survival Data*. Springer-Verlag, New York.

Graphical Techniques

Chambers, J.M., Cleveland, W.S., Kleiner, B., and Tukey, P.A. (1983). *Graphical Methods for Data Analysis*. Duxbury Press, Belmont, CA.

Cleveland, W.S. (1993). *Visualizing Data*. Hobart Press, Summit, NJ.

Cleveland, W.S. (1994). *The Elements of Graphing Data*. Hobart Press, Summit, NJ.

S-PLUS Programming

Becker, R.A., Chambers, J.M., and Wilks, A.R. (1988). *The New S Language*. Wadsworth & Brooks/Cole, Pacific Grove, CA.

Chambers, J.M. (1998). *Programming with Data*. Springer-Verlag, New York.

Venables, W.N. and Ripley, B.D. (2000). *S Programming*. Springer-Verlag, New York.

TYPOGRAPHIC CONVENTIONS

Throughout this supplement, the following typographic conventions are used:

- `This font` is used for S-PLUS expressions and code samples.
- **This font** is used for elements of the S-PLUS user interface, for operating system files and commands, and for user input in dialog fields.
- *This font* is used for emphasis and book titles.
- CAP/SMALLCAP letters are used for key names. For example, the Shift key appears as SHIFT.
- When more than one key must be pressed simultaneously, the two key names appear with a hyphen (-) between them. For example, the key combination of SHIFT and F1 appears as SHIFT-F1.
- Menu selections are shown in an abbreviated form using the arrow symbol (►) to indicate a selection within a menu, as in **File ► New**.

CONNECT/C++ 2

Introduction **20**
Resources 20

Simple Examples: An Application and a Called Routine **21**
Creating a Simple Application 21
Example of Calling a C Function Via `.Call` 23

CONNECT/C++ Class Overview **28**
Data Object Classes 28
Function Evaluation Classes 28
Client-to-Engine Connection Classes 29
Evaluator Classes 29

CONNECT/C++ Architectural Features **31**
`CSPobject` 31
Constructors and Generating Functions 31
Constructing From an Existing Object 32
Assignment Operators 33
Overloading Operators 33
Converting C++ Objects to S-PLUS Objects 34
Subscripting Operators 35
Subscript and Replacement Operations 36
Subscript and Arithmetic Operations 36
Matrix Computations 37
Printing to Standard Output 38
Named Persistent Objects 38
Storage Frames For Unnamed Objects 40

INTRODUCTION

CONNECT/C++ is a tool used for interfacing C++ with the S language. It is a convenient tool for integrating the S-PLUS engine inside other programs written in C++, but it can also be used for integrating C++ code into the S-PLUS environment.

To enable communication between the GUI (Graphical User Interface) and S-PLUS, CONNECT/C++ was developed to provide a framework for the S language version 4-based engine used in S-PLUS 6.0 for Windows. In fact, the S-PLUS 6 GUI provides the most comprehensive example of using CONNECT/C++ to integrate the S-PLUS engine with C++ applications. Similarly, C++ developers could create their own GUI to interface with S-PLUS using the same technique.

CONNECT/C++ is a class library providing C++ classes with member functions that operate on S-PLUS objects similar to S methods in the S language. Users can use these classes and their member functions to create and manipulate persistent as well as local S objects.

CONNECT/C++ provides various mechanisms for evaluating S expressions inside a C++ program and module. S-PLUS 6.*x* ships with several examples that illustrate how to use this library. Some of these examples contain pairs of equivalent S and C++ functions that perform the same tasks. The speed of the C++ functions can be many times faster than the S code depending on the code's complexity and the data sizes. The examples are located in the **SHOME/sconnect** directory, where **SHOME** is your S-PLUS installation directory.

Resources

For more information on CONNECT/C++, go to **SHOME/sconnect/help/ConnectC++.Class.library.htm**. This HTML file is a guide to the CONNECT/C++ class library for C++ developers, and it discusses how to connect to the S-PLUS engine, how to create data objects, call S-PLUS functions, and evaluate S-PLUS syntax.

SIMPLE EXAMPLES: AN APPLICATION AND A CALLED ROUTINE

CONNECT/C++ can be used for two distinct purposes: to create C++ applications that can access S-PLUS functionality, and to create C++ functions that can be called via the S-PLUS `.Call` interface. We begin our investigation of CONNECT/C++ with a simple example of each.

Creating a Simple Application

The CONNECT/C++ application used in this example is a console application that creates two S-PLUS vectors. It then uses S-PLUS to compute a linear model relating the two vectors.

The code begins with the inclusion of **sconnect.h**, the CONNECT/C++ library which all CONNECT/C++ code must reference at the start. It then declares a global S-PLUS connection object, with the CONNECT/C++ class `CSPengineConnect`, before beginning the main application function. The `CSPengineConnect` class generates a connection between the client application and S-PLUS, allowing you to create S-PLUS objects in the permanent frame, notifying you when the databases are attached or detached to the client, and evaluating S language expressions. Here's what the code looks like so far:

```
#include "sconnect.h"

// A global connection object
CSPengineConnect g_engineConnect;

int main(int argc, char* argv[])
{
```

The first step in the main function is to create the actual connection object, which opens a connection to S-PLUS:

```
// Create the connection to S-PLUS
g_engineConnect.Create( argc, argv);
```

We then create the variables `x` and `y` to use in the regression. The CONNECT/C++ class `CSPnumeric` is used to store S-PLUS numeric vectors. The `CSPnumeric` class is one of many in CONNECT/C++ that are used to represent S-PLUS objects within C++. Similar classes

exist for most of the standard atomic objects in S-PLUS (see Table 2.1). The `Create` method creates instances of the class; the `Assign` method assigns the class to an S-PLUS database:

```
// Create S object with name "x" in the current database.
// Same as x<-1:10 at the command line.
CSPnumeric sx;
sx.Create("1:10","x");

// Squaring sx, which is the same as S expression
// sy <- x*x in a local frame, but here we set it to local
// C++ variable sy.

CSPnumeric sy = sx * sx;

// Assign the result as S object with name "y" in the
// current database.
sy.Assign("y");
```

Finally, we fit the linear model, passing the appropriate call to S-PLUS via the CONNECT/C++ method `SyncParseEval`:

```
  // Evaluate z<-lm(y~x)
  g_engineConnect.SyncParseEval("z<-lm(y~x)");

  return 1;
}
```

The complete code for this example is in the directory **SHOME/sconnect/samples/splm**, and the C++ code is in the file **splm.cxx**.

Run the application as follows:

1. Change the current directory to the directory containing the code:

   ```
   cd SHOME/sconnect/samples/splm
   ```

   where **SHOME** is your S-PLUS installation directory.

2. Build the program:

   ```
   Splus6 CHAPTER -sconnectapp *.cxx
   Splus6 make
   ```

3. Run the program:

```
Splus6 EXEC S.app
```

To verify the results, start S-PLUS:

```
Splus6
S-PLUS : Copyright (c) 1988, 2002 Insightful Corp.
S : Copyright Lucent Technologies, Inc.
Version 6.1.2 Release 1 for Sun SPARC, SunOS 5.6 : 2002
Working data will be in .Data
```

and look at the objects `x`, `y`, and `z`:

```
> x
[1]  1  2  3  4  5  6  7  8  9  10
> y
[1]  1  4  9  16  25  36  49  64  81  100
> z
Call:
lm(formula = y ~ x)

Coefficients:
 (Intercept)  x
         -22 11

Degrees of freedom: 10 total; 8 residual
Residual standard error: 8.124038
```

Example of Calling a C Function Via `.Call`

The Gauss-Seidel method is a familiar technique for solving systems of linear equations. The algorithm is straightforward and easy to implement in S-PLUS:

```
gaussSeidel<-
# gaussSeidel solves a linear system using Gauss-Seidel
# iterative method.
# REQUIRED ARGUMENTS:
#      A and b are numeric matrix and vector respectively.
# VALUE:
#      a vector x, solution of A x = b
#
```

```
# Usage:
#  A<-matrix(rnorm(100),nrow=10)
#  diag(A)<-seq(ncol(A),ncol(A)) #Make it diagonally
#                                # dominant
#  b<-rnorm(ncol(A))
#  sys.time({x1<-gaussSeidel(A,b)})
function(A,b)
{
  # Hard-coded relative tolerance and max iterations
  tol<-1.0e-4
  maxItr<-1e4

  # Validating
  A <- as.matrix(A)
  b <- as.numeric(b)
  if(nrow(A)!=ncol(A) || ncol(A)!=length(b))
     stop("nrow(A)!=ncol(A) || ncol(A)!=length(b)")

  # Begin Gauss-Seidel step
  x<-b
  for(k in 1:maxItr)
  {
     xOld<-x
     for(i in 1:nrow(A))
     {
        s<- A[i,i]*x[i]
        for(j in 1:ncol(A))
           s <- s - A[i,j]*x[j]
        x[i] <- (b[i]+s)/A[i,i]
     }
     # Check convergence; continue if necessary
     if(max(abs((x-xOld)/x)) < tol)
        return(x);
  }
  warning("Solution does not converge\n")
  return(x)
}
```

This code, which involves a nested loop, could be made more efficient, but the intention is to illustrate the Gauss-Seidel iteration in its most familiar form. An example including the implementation of

CONNECT/C++ is shown below, and notice that by using the classes and methods of CONNECT/C++, this code closely resembles the equivalent computation in S-PLUS.

The code begins by including the **sconnect.h** header file to give us access to the CONNECT/C++ library. Next, it includes the header file required for the Gauss-Seidel code itself:

```
# include "sconnect.h"
# include "gausssdl.h"
```

We then declare the `gaussSeidel` object as an object of class `s_object`, as required by the `.Call` interface:

```
s_object* gaussSeidel(s_object* ps_A, s_object* ps_b)
```

As is typical for S-PLUS code, we declare the `S_EVALUATOR` and then embed the implementation in a `try-catch` block. Within the `try` block, the tolerances are hard-coded. We then construct the C++ equivalents to the S-PLUS objects `A` and `b`:

```
{
  S_EVALUATOR
  try
  {
    // Hard-coded relative tolerance and max iterations
    double tol =1e-4;
    long maxItr = 1000;

    // Constructing and validating C++ objects
    CSPnumericMatrix A(ps_A);
    CSPnumeric b(ps_b);
    if(A.nrow()!=A.ncol() || A.ncol()!=b.length())
      PROBLEM "A.nrow()!=A.ncol() || A.ncol()!=b.length()"
       ERROR;
```

The actual Gauss-Seidel step follows:

```
    // Begin Gauss-Seidel step
    CSPnumeric x=b;
    for(long k =1; k<= maxItr; k++)
    {
      CSPnumeric xOld = x;
      for(long i= 1; i <= A.nrow(); i++)
```

```
        {
          double s = A(i,i) * x(i);
          for(long j = 1; j <= A.ncol(); j++)
            s  = s - A(i,j) * x(j);
          x(i) = (b(i)+s)/A(i,i);
        }
        // Check convergence; continue if necessary
        if(Max(abs((x-xOld)/x)) < tol)
            return(x);
      }
      PROBLEM "Solution does not converge" WARN;
      return(x);
    }
    catch(...)
    {
    }
    return(blt_in_NULL); // return the built-in NULL object
  }
```

The complete code for this example is in the directory **SHOME/sconnect/samples/gausssdl**, with the C++ code in the file **gausssdl.cxx**.

To compile and execute the C++ code:

1. Change the current directory to the directory containing the code:

   ```
   cd SHOME/sconnect/samples/gausssdl
   ```

2. Build the share library:

   ```
   Splus6 CHAPTER -sconnectlib *.cxx
   Splus6 make
   ```

3. Run S-PLUS:

   ```
   Splus6
   ```

With the makefile created by `CHAPTER`, compiling your code is simple: just run the `make` command as an S-PLUS utility as shown in step 2.

The `Splus6` in front of `make` allows S-PLUS to set its environment variables appropriately before calling the standard make utility; in particular it defines the **SHOME** environment variable used in the makefile.

The `make` utility executes the necessary commands to compile and link the C++ code into the shared object `S.so`. Note that `-sconnectlib` is required to include the CONNECT/C++ library.

The CONNECT/C++ called via `.Call` implementation runs considerably faster than the S-PLUS code. Here is a comparison for a matrix `A` with 100 columns and 100 rows on a Solaris machine:

```
> A<-matrix(rnorm(10000),nrow=100); diag(A)<-seq(ncol(A),
+ ncol(A)) # Make it diagonally dominant
> b<-rnorm(100);
> sys.time({x1<-gaussSeidel(A,b)})
[1] 37.00 39.35
> sys.time({x2<-.Call('gaussSeidel',A,b)})
[1] 0.04 0.04
```

The CONNECT/C++ version ran about 1000 times faster than the pure S-PLUS version!

CONNECT/C++ CLASS OVERVIEW

The class library provides a set of classes that can be used to create and manipulate persistent data objects, run S-PLUS functions, parse and evaluate S-PLUS expressions, and receive output and notification when objects are changed or when databases are attached and detached.

The following sections provide an overview of specific categories of classes used to accomplish these operations.

Data Object Classes

Data object classes provide methods to create and operate on arrays, matrices, and vectors. To use these classes to create a data object, simply call the object constructor or call the `Create()` method. For a persistent object, specify the name of the object and an S language expression you want to parse, evaluate, and assign the result in order to initialize it with data. Alternatively, a data object can be constructed using a form of the constructor that takes an optional S language expression as an argument. This is useful if named (persistent) objects are not required, but intialization is required. Once the object is created, methods can be used to operate on the object.

To receive notification in a client application when a data object changes, create a new class in the client application derived from the appropriate base class and override the virtual methods for handling object notification. When a named object is modified or removed, those virtual methods in the client are called.

Function Evaluation Classes

The `CSPcall` class allows S-PLUS functions to be evaluated with arguments passed to the function. Arguments are any `S_object` as well as objects derived from `CSPobject`, which may include data objects and other S-PLUS objects. Results are returned as a `CSPobject` to the client. To use this class, simply call the object constructor with the name of the function to run and any arguments you wish to pass from the client to the function.

Client-to-Engine Connection Classes

The `CSPengineConnect` class creates a connection between the client and the S-PLUS engine. This connection permits creation of objects in the permanent frame, creation of persistent unnamed objects outside of `.Call` routines, notification in the client when databases are attached or detached, output routing to the client, and evaluation of S language expressions.

To use `CSPengineConnect`, create a new class derived from `CSPengineConnect` in the client, override the virtual methods for receiving database attach/detach notification, and output notification, and add a member variable to the client application class object to record a reference to a single instance of this derived class.

Use of the `CSPengineConnect` class is only necessary when one or more of the following features is desired in the client program:

- Integrate S+ engine shared library **libSqpe.so** with another application (client).
- Notification in the client when databases are attached or detached and when changes are made in persistent objects.
- Output redirected to the client.

For more information on using `CSPengineConnect`, please see the section on this class in **SHOME/sconnect/help/ConnectC++.Class.library.htm**.

Evaluator Classes

The `CSPevaluator` class manages memory resources, errors, the top-evaluation frame, and a set of local evaluation frames. Although it is optional, instantiating an object of `CSPevaluator` class at the top of a `try` block can speed up the code, and the corresponding `catch` block receives an exception error when an unexpected error occurs in the S-PLUS engine.

To use `CSPevaluator`, create an instance of this class at the top of a `try` block as shown below:

```
double minValue = 0;
try
{
      // Open top-level-evalutor (frame 1) if it is closed
      CSPevaluator sEvaluator;
      CSPnumeric myNumeric = sEvaluator.eval("1:10");
      minValue = myNumeric.Min(); //minValue = 1
```

```
} // Close top-level evaluator when sEvaluator is out of
  // scope
catch(...)
{
      // Unexpected error occurred in the engine
}
```

For more information on using `CSPevaluator`, please see the section on this class in **SHOME/sconnect/help/ConnectC++.Class.library.htm**.

CONNECT/C++ ARCHITECTURAL FEATURES

The following sections describe the basic architectural featues in the class library and some of the specific programming features available in the library that make it possible to perform S-PLUS operations efficiently in client programs and modules written in C++. Classes and methods discussed in this section are fully documented in the reference sections for the classes in the online help for CONNECT/C++.

CSPobject

`CSPobject` is the base class of most of the classes that represent S-PLUS classes. It provides common functionality to its derived classes, and its most important data member is:

```
s_object* CSPobject::m_ps_object
```

A class that represents an S-PLUS class inherits `m_ps_object` because `CSPobject` is its base class. As a smart pointer, a derived class of `CSPobject` provides safer methods to manipulate the data pointed by `m_ps_object` as compared to using global C functions. For example, the constructor, the destructor, and the assignment operators automatically increment and decrement reference counts whenever appropriate to provide the same data sharing mechanism as that of the SV4 language.

All `CSPobject`-derived classes have a method called `IsValid()` which allows you to test whether the member `m_ps_object` is valid or not.

Constructors and Generating Functions

Often, S generating functions are more convenient than the S method `new`. Similarly, constructors of CONNECT/C++ classes can provide the same convenience. They have the following form:

```
CSPclass::CSPclass(const char* pszExpression);
    // pszExpression is a string representing valid S code.
```

where `class` is a `CSPobject`-derived object.

This form of the object constructor parses and evaluates `pszExpression` and uses the resultant S-PLUS object as its value. Normally, `pszExpression` should contain an S-PLUS expression that calls to an appropriate generating function. However, it works for any S-PLUS expression that returns a valid S-PLUS object, and the

constructor automatically coerces the returned object to the class that it represents. It increments the reference count upon completion, as well. In case of errors, the constructor throws an exception in the client application.

For example:

```
CSPevaluator s;

CSPinteger x("1:4");              // x<-1:4
CSPnumeric y("fuel.frame[,1]");   // y<-as(fuel.frame[,1],
                                  //   'numeric')
CSPnumeric z("new('numeric')");   // z<- new('numeric')
CSPmatrix A("matrix(1:4, nrow=2)"); // A<-matrix(1:4,
                                    //     nrow=2)
CSPmatrix B("1:4");               // B<-as(1:4,'matrix')

// Do something with x,y,z,A, and B
```

Constructing From an Existing Object

You can construct new objects from existing objects using one of the following forms:

```
CSPclass::CSPclass(const CSPclass& sObject);//copy
                                            //constructor
CSPclass::CSPclass(s_object* ps_object);    //construct
                                            //from s_object
```

where `class` is a `CSPobject`-derived object.

The copy constructor of a CONNECT/C++ class behaves like an S-PLUS assignment operator when the S-PLUS object name is first used. They both share the same data with the object names used to construct them. However, for the CONNECT/C++ classes, sharing is not possible if the classes are incompatible. It increments the reference count upon completion.

An example of creating new objects from existing objects follows:

```
CSPevaluator s;

CSPnumeric x("1:4"); // x<-1:4
CSPnumeric u(x);     // u<-x # u shares data with x
CSPmatrix A(x); // A<-as(x,'matrix') # A shares data with x
```

```
CSPcharacter v(x);   // v<-as(x,'character') # no sharing

s_object* ps_object = x.GetPtr();//Get pointer to s_object*
CSPnumeric U(ps_object);         // U shares data with x
CSPmatrix a(ps_object);          // a shares data with x
```

Assignment Operators

The assignment operator of an CONNECT/C++ class behaves like an S-PLUS assignment operator when the S-PLUS object name is already used. However, the left-hand-side object of the operator = is an existing and valid object. The assignment operator decrements the reference count on the old object and increments the reference count on the new object before swapping the two object pointers:

```
CSPclass& CSPclass::operator=(const CSPclass& sObject);
```

where `class` is a `CSPobject`-derived object.

An example of the assignment operator follows:

```
CSPevaluator s;

CSPnumeric x("1:4"); // x<-1:4
CSPnumeric u = x;    // u<-new('numeric'); u<-x # u shares
                     // data with x
CSPmatrix A = x;    // A<-new('matrix'); A<-as(x,'matrix')
                    // # no sharing

CSPnumeric y;  // y<-new("numeric")
u = y;       // u<-y # u switches to share data with y
A = y;//A<-as(y,'matrix') # A switches to share data with y
```

Overloading Operators

CONNECT/C++ contains some useful overloading operators such as +, -, * and /. These operators perform element-by-element operations in the same way as in the S language. However, for the matrix class, the * operator is different. The operator for `CSPmatrix` is a real matrix multiplication operator equivalent to the S `%*%` operator.

```
CSPclass& CSPclass::operator+(const CSPclass& sObject);
CSPclass& CSPclass::operator-(const CSPclass& sObject);
CSPclass& CSPclass::operator*(const CSPclass& sObject);
CSPclass& CSPclass::operator/(const CSPclass& sObject);
```

where `class` is a `CSPobject`-derived object.

An example using the CSPmatrix follows:

```
CSPevaluator s;

CSPnumeric x("1:4"); // x<-1:4
CSPnumeric y ("4:1"); // y<-4:1
y = y+x*x;    // y<-y+x*x

CSPmatrix A("matrix(1:4,nrow=2)");//A <- matrix(1:4,nrow=2)
CSPmatrix B("matrix(4:1,nrow=2)");//B <- matrix(4:1,nrow=2)
CSPmatrix D = A*A + B*B;          //D <- A %*% A + B %*% B
```

Converting C++ Objects to S-PLUS Objects

Objects derived from class CSPobject are C++ representations of S-PLUS objects; within S-PLUS, S-PLUS objects are represented as C objects of type s_object*. Sometimes, an application needs to access the s_object* directly. For example, the arguments and the return value of all .Call interfaces must be of type s_object*.

The CSPobject class provides a convenient way to automatically convert to s_object*. Simply use a CSPobject wherever a s_object* is required. It automatically invokes a conversion operator that returns the s_object* as appropriate.

```
s_object* CSPobject::operator*();
s_object* CSPobject::operator&();
```

For example:

```
s_object* myCall()
{
        CSPnumeric x("1:10");
        return x;
}
s_object *pReturn = myCall();
```

The return statement, return x, first typecasts x to type s_object*. This invokes the conversion operator s_object *() of the CSPnumeric class (derived from CSPobject) which ensures that the destructor of x does not delete the object, even if the reference count drops to zero.

Subscripting Operators

CONNECT/C++ contains some useful overloading subscripting operators `()` for the derived classes of `CSPvector` and `CSParray` such as `CSPnumeric` and `CSPmatrix`. The proxy class of the returned object provides supports for read/write and mixed-mode operations:

```
const double CSPnumeric::operator()(long lIndex); const
      // Fortran style indexing starting from index 1
      // rvalue only

CSPproxy CSPnumeric::operator()(long lIndex);
      // Fortran style indexing and ordering
      // lvalue and rvalue
```

An example using the subscripting operators:

```
CSPevaluator s;

CSPnumeric x("c(0.1, 0.2, 0.8, 0.9)"); // x<- c(0.1, 0.2,
                                       //       0.8, 0.9)
double d = x(1);   // d <-x[1] # d is 0.1
d = d + x(2);      // d<- d+x[1] # d is 0.3
double e = (long) x(1);  // e<-as.integer(x[2]) # e is 0
long n = x(1);           // n <-as.integer(x[1]) # n is 0
n = n + x(2);   // n <- n+as.integer(x[2]) # n is still 0
```

The following is another example using the subscripting operator for a matrix:

```
CSPevaluator s;

CSPmatrix A("matrix(c(0.1, 0.2, 0.8, 0.9),  2)");
       // A<- matrix(c(0.1, 0.2, 0.8, 0.9),  2)

double d = A(1,1);         // d <-A[1,1] # d is 0.1
d = d + A(2,1);            // d<- d+A[2,1] # d is 0.3
long e = (long) A(2,1); // e<-as.integer(A[2,1]) # e is 0
long n = A(1,1);       // n <-as.integer(A[1,1]) # n is 0
n = n + A(2,1); //n <- n+as.integer(A[2,1]) # n is still 0
```

Subscript and Replacement Operations

If a subscript operator of a `CSPobject`-derived class returns an `lvalue` object of `CSPproxy`, the operation involves replacing an element of the S-PLUS object. Since writing data is not possible for a shared S-PLUS object, `CSPproxy` must determine whether to copy data before replacing its elements. This action occurs in one of its overloaded assignment operations:

```
CSPproxy& CSPproxy::operator=(long);
CSPproxy& CSPproxy::operator=(double);
CSPproxy& CSPproxy::operator=(const CSPproxy&);
```

For example:

```
CSPevaluator s;

CSPnumeric x("1:4");        // x<- 1:4
x(1) = 0.0;                 // x[1]<- 0 # x is not share,
                            // simply set x[1] to 0.0
x(2) = x(1);       // x[2]<- x[1] # x is not share, simply
                   // set x[2] to 0.0
CSPnumeric y(x);   // y<- x # y shares data with x
y(1)= 10.0;        // y[1]<- 10 #copy and replace:
                   // y[1] is 10 and x[1] is 0
```

Subscript and Arithmetic Operations

Some overloaded operators are available to support mixed-mode arithmetic operations involving subscripting objects of classes derived from `CSPobject`. These operators, `+`, `-`, `*` and `/`, perform mixed-mode operations following the same rules as S-PLUS:

```
long CSPproxy::operator+(long)
double CSPproxy::operator+(double)
...
```

An example using the arithmetic operators:

```
CSPevaluator s;

CSPnumeric x("1:4"); // x<- 1:4
CSPnumeric y(x);     // y<- x # y shares data with x
// A <- matrix(1:4,nrow=2)
CSPmatrix A("matrix(1:4,nrow=2)");
```

```
// e <- A[1,1] + A[1,2]
double e = A(1,1)+A(1,2);
// A[1,2] <- e*(A[1,1]+A[2,1])
A(1,2) = e*(A(1,1)+A(2,1));
// A[2,2] <- x[1]*A[1,1]+y[2]*A[2,1]
A(2,2) = x(1)*A(1,1)+y(2)*A(2,1);

// X<-array(1:16, c(2,2,2,2))
CSParray X("array(1:16, c(2,2,2,2))");
// X[1,1,1,1] <- X[2,1,1,1]+e;
X(1,1,1,1) = X(2,1,1,1) + e;
// X[2,1,1,1] <- y[1] - X[2,1,1,1];
X(2,1,1,1) = y(1) - X(2,1,1,1);
// X[1,2,1,1] = A[1,1] * X[2,1,1,1];
X(1,2,1,1) = A(1,1) * X(2,1,1,1);
```

Matrix Computations

Some overloaded functions are available for matrix computations, such as the example below.

```
double CSPmatrix::ConditionNumber(void);
CSPmatrix SPL_Multiply(const CSPmatrix& A,
     const CSPmatrix& B);
CSPnumeric SPL_Multiply(const CSPmatrix& A,
     const CSPnumeric& x);
...
```

For example:

```
CSPevaluator s;

CSPmatrix A("matrix(5:8, nrow=2)");
 // A<- matrix(5:8, nrow=2)
CSPmatrix B(A);                    // B<- A
CSPmatrix D = SPL_Multiply(A, B); // D<-A %*% B

CSPnumeric x("1:2");                // x<- rnorm(2)
CSPnumeric y = SPL_Multiply(A, x); // y<- A %*% x
```

Printing to Standard Output

You can use the following CONNECT/C++ method to print to the S-PLUS standard output stream:

```
void CSPobject::Print(void);
```

For example:

```
CSPevaluator s;

CSPcharacter message("'hello'");  //message <- 'hello'
message.Print();                  //print(message)

CSPmatrix M("matrix(1:4,nrow=2)");//M<-matrix(1:4, nrow=2)
M.Print();                        //print(M)
```

Named Persistent Objects

All `CSPobject`-derived objects are placeholders for an `s_object` that exists in the engine. So, this C++ object can reference an `s_object` or none at all, depending on whether the member `s_object` pointer points to a valid `s_object`. All `CSPobject`-derived classes have a method called `IsValid()` which allows you to test whether it is pointing to a valid `s_object` or not.

All named objects are created in a permanent frame associated with an S-PLUS database, and are thus persistent between calls and between sessions in the S engine. When you create a new `CSPobject` in your client program, a new `s_object` is created in the S engine. When you delete this `CSPobject`, the `s_object` is also released in the engine. However, when you execute S-PLUS expressions to remove the `s_object` that your `CSPobject` points to, such as by using `rm(myObject)`, or you call the `Remove()` method on the object, the `CSPobject` is not deleted in your client. The `OnRemove()` method of the `CSPobject` in your client is called and the base class version of this method "disconnects" your `CSPobject` from the now released `s_object` by setting the member `s_object` pointer to `NULL`. After this event, calling `IsValid()` on the `CSPobject` returns `FALSE`.

Deleting the `CSPobject` in your client program does not automatically remove the permanent frame `s_object` in the S-PLUS engine that this `CSPobject` refers to. You must call the method `Remove()` to remove the `s_object` from the engine.

You can create named objects using the `Create()` method of the various object classes derived from `CSPobject`, such as `CSPnumeric`. Whenever these objects are modified, the `OnModify()` method is

called in your client program. Whenever these objects are removed, the `OnRemove()` method is called in your client program. Only named objects support this kind of client program notification.

To create a named object in your client, first derive a new class from the appropriate `CSPobject`-derived class, such as `CSPnumeric`. Then, construct an instance of this derived class using the constructor, then call the `Create()` method to specify the name you wish to give the object. It is important to derive a new class from the `CSPobject`-derived class instead of just using the base class directly in your client because the `OnModify()` and `OnRemove()` methods are virtual and must be overridden in your derived class in the client in order to be notified when these events occur.

A `CSPobject` can be modified in one of two ways. It can be modified in the client program by using the operators available for the object to assign and operate on the elements of the object. When this kind of modification is done, it is necessary to call the `Commit()` method on the object to commit it to the S-PLUS engine before any changes to the object are reflected in the persistent `s_object` that is referenced by the object in the client.

Another way it can be modified is by evaluating S-PLUS expressions, such as by using `CSPengineConnect::SyncParseEval()`. When this kind of modification is done, it is not necessary to call `Commit()` on the object, as the `s_object` is automatically updated by the S-PLUS engine. For both kinds of modification, the `OnModify()` method of the `CSPobject` is called in the client program. It is important to call the base class `OnModify()` in your override of `OnModify()`. This allows the base class to update the member `s_object` pointer to point to the newly modified `s_object` in the engine.

The `s_object` member of a `CSPobject` can be removed (invalidated) in one of two ways:

1. It can be removed in the client program by calling the `Remove()` method on the `CSPobject`. This method removes the `s_object` from the permanent frame and triggers a call to the `OnRemove()` method of the `CSPobject` in the client program. The base class version of `OnRemove()`, which should be called at the end of the overridden version in the client, releases the member `s_object` from the `CSPobject`.

2. It can be removed by evaluating S-PLUS expressions, such as by calling `CSPengineConnect::SyncParseEval()`. This also triggers a call to the `OnRemove()` method of the `CSPobject` in the client program.

Storage Frames For Unnamed Objects

Normally, when you create an unnamed `CSPobject` in a client routine that you call via `.Call`, the `s_object` corresponding to this `CSPobject` is "alive" or is valid until the routine ends and scope changes out of the routine.

If you create an unnamed `CSPobject` when the S-PLUS evaluator is not open, the `s_object` corresponding to this `CSPobject` may not be valid. For most client applications, this is usually inadequate. Therefore, you need to do the following to ensure that an unnamed `CSPobject` created in a client application does not get released until the end of the client routine:

- Create an instance of a `CSPevaluator` at the top of the scope "`{`."
- Create and use any unnamed `CSPobject`-derived objects in the client.

For example:

```
{
        CSPevaluator s;
        CSPnumeric x("1:10");
        ...
}
```

For named objects, you do not have to use the above approach: simply create named `CSPobject`-derived objects using the constructor and a call to `CSPobject::Create()`. For further information, see the online help for the classes `CSPengineConnect::OpenTopLevelEval()`, `CSPengineConnect::CloseTopLevelEval()`, and the `Create()` method for the object type to be created.

Table 2.1: *CONNECT/C++ classes and their S-PLUS counterparts.*

S-PLUS Class	CONNECT/C++ Class	Example
any	CSPobject	CSPobject x("2")
numeric	CSPnumeric	CSPnumeric x("2.32")
integer	CSPinteger	CSPinteger x("2")
logical	CSPlogical	CSPlogical x("c(T,F")
character	CSPcharacter	CSPcharacter("abcd")
named	CSPnamed	CSPnamed("c(a=1,b=2, d=3)")
matrix	CSPmatrix CSPnumericMatrix CSPcharacterMatrix	CSPmatrix A("matrix(1:4,2)") CSPnumericMatrix A("matrix(rnorm(12,6") CSPcharacterMatrix A("matrix(letters[1:12],6)")
array	CSParray	CSParray B(“array(1:8,c(2,2,2))"
list	CSPlist	CSPlist("list(1:2,6:700)")
function	CSPfunction	CSPfunction ("function(x) x^2")
call	CSPcall	CSPcall("lm")

GLOBALIZATION 3

Introduction **44**

Working With Locales **45**
- Setting a Locale 46
- Changing a Locale 46
- Changing the Collation Sequence 46

Using Extended Characters **48**
- In Variable Names 48
- In PostScript Output 49

Importing, Exporting, and Displaying Numeric Data **50**
- Importing and Exporting Data 50
- Displaying Data 51

INTRODUCTION

New in S-PLUS 6.1 are a number of enhancements designed to improve global installation and use:

- New functions have been added that allow you to specify a locale to be used by S-PLUS. Specifically, S-PLUS 6.1 now supports French and German locales.
- Sorting and collating functions have been modified to support different locales.
- The use of 8-bit characters (Latin1/Western European character set) has been enabled. The 8-bit characters can now be used in variable names and will properly display in data.
- Functions that import, export, and display numeric data have been modified to support different locales.
- S-PLUS Setup has been improved to test for local settings and install localized DLLs as necessary.

In the sections that follow, we describe these enhancements in greater detail.

WORKING WITH LOCALES

A locale determines which characters are considered printable or alphabetic, the format for numerals, and the collating sequence of characters.

Note

The implementation of locales in S-PLUS does not cover date formats (date formats are handled through a separate mechanism) and messaging (all messages in S-PLUS are in the English language).

The default locale is `C`, the locale used by the C language (7-bit ASCII characters and U.S.-style numbers). The `C` locale conforms to the behavior of earlier versions of S-PLUS, that is:

- No accented characters are displayed (backslash octal instead).
- The decimal marker is a period.
- The sort order is all uppercase followed by all lowercase.

In European-based locales, however, S-PLUS prints accented characters and uses a more natural sort order ("`a`", then "`A`", then "`à`" (`"\340"`), then "`À`" (`"\300"`)). In continental-based locales, the decimal marker is a comma. In English-based (and French Canadian) locales, the decimal marker is a period.

Not every locale is supported on every computer. To see which locales are supported on a computer running a version of UNIX, run `locale -a`.

Hint

Choose a locale with "8859-1" in the name if there is a choice between this and "Roman8," as S-PLUS assumes you are using the ISO 8859-1 character set. Your terminal emulator should be using an 8859-1 font. (This is an issue on HPUX where the default seems to be "Roman8.")

Setting a Locale

Setting a locale gives you a way to tell S-PLUS which character sets, number formats, and collation sequences to use. The primary function for setting a locale is the `Sys.setlocale()` function. This function allows you to set `LC_ALL`, `LC_COLLATE`, `LC_CTYPE`, `LC_MONETARY`, `LC_NUMERIC`, or `LC_TIME`.

To use the default regional settings for locale, type the following:

```
> Sys.setlocale(locale="")
```

This command tells S-PLUS to follow the UNIX environment variables (`LANG`, `LC_{CTYPE,COLLATE,NUMERIC}`).

Hint

If you put the desired `Sys.setlocale()` command into your **S.init** file, the command will be run automatically each time you launch S-PLUS.

Changing a Locale

Use the `Sys.setlocale()` and `Sys.getlocale()` functions to set and get locales, respectively. Use the `Sys.withlocale()` function if you want to evaluate an expression within a specified locale.

For example:

```
> Sys.getlocale()                    # Default
[1] "C"
> Sys.setlocale(locale="en_US")
[1] "en_US"
> Sys.getlocale()
[1] "en_US"
```

Changing the Collation Sequence

"Collation sequence" refers to the ordering of characters by the `sort()` and `order()` functions. For example, in the `C` locale, uppercase (capital) letters collate before lowercase letters. In almost all other locales, however, collation sequence ignores case.

For example:

```
> Sys.setlocale(locale="C")
[1] "C"
> sort(c("as", "Axe"))
[1] "Axe" "as"
```

```
> Sys.setlocale(locale="English (United States)")
[1] "English_United States.1252"
> sort(c("as", "Axe"))
[1] "as"  "Axe"
```

Changing `LC_COLLATE` affects the way S-PLUS interprets the ordering of character data and thus can be used to change the behavior of functions like `sort()` and `order()`.

USING EXTENDED CHARACTERS

S-PLUS 6.1 accepts more "alphabetic" characters than in earlier versions. This set of "alphabetic" characters is fixed and is the set of characters considered alphabetic in ISO 8859-1 (Latin1/Western European character set).

The set of characters that are displayed as octal codes changes depending on the current locale (specifically, `LC_CTYPE`). For example, in all locales that use Latin characters, `"\341"` is displayed as "á". This affects how functions such as `format()`, `print()`, and `cat()` display character data.

In Variable Names

Because S-PLUS now supports Western European character sets (ISO 8859-1), you can include 8-bit ASCII characters (ASCII codes 128-255) within character data and within the names of S-PLUS objects. For example, you can include names such as Furtwängler and García Íñiguez in your character data and they will properly display. S-PLUS uses your locale settings to determine the appropriate character set to use in printing output.

The characters allowed in S-PLUS object names are:

```
alphanumerics <- c(".", "0", "1", "2", "3", "4", "5", "6",
  "7", "8", "9", "A", "B", "C", "D", "E", "F", "G", "H",
  "I", "J", "K", "L", "M", "N", "O", "P", "Q", "R", "S",
  "T", "U", "V", "W", "X", "Y", "Z", "a", "b", "c", "d",
  "e", "f", "g", "h", "i", "j", "k", "l", "m", "n", "o",
  "p", "q", "r", "s", "t", "u", "v", "w", "x", "y", "z",
  "\212", "\214", "\216", "\232", "\234", "\236", "\237",
  "\300", "\301", "\302", "\303", "\304", "\305", "\306",
  "\307", "\310", "\311", "\312", "\313", "\314", "\315",
  "\316", "\317", "\320", "\321", "\322", "\323", "\324",
  "\325", "\326", "\330", "\331", "\332", "\333", "\334",
  "\335", "\336", "\337", "\340", "\341", "\342", "\343",
  "\344", "\345", "\346", "\347", "\350", "\351", "\352",
  "\353", "\354", "\355", "\356", "\357", "\360", "\361",
  "\362", "\363", "\364", "\365", "\366", "\370", "\371",
  "\372", "\373", "\374", "\375", "\376", "\377")
```

The characters not allowed in names are:

```
non.alphanumerics <- c("\001", "\002", "\003", "\004",
  "\005", "\006", "\007", "\b", "\t", "\n", "\013", "\014",
  "\r", "\016", "\017", "\020", "\021", "\022", "\023",
  "\024", "\025", "\026", "\027", "\030", "\031", "\032",
  "\033", "\034", "\035", "\036", "\037", " ", "!", "\"",
  "#", "$", "%", "&", "'", "(", ")", "*", "+", ",", "-",
  "/", ":", ";", "<", "=", ">", "?", "@", "[", "\\", "]",
  "^", "_", "`", "{", "|", "}", "~", "\177", "\200",
  "\201", "\202", "\203", "\204", "\205", "\206", "\207",
  "\210", "\211", "\213", "\215", "\217", "\220", "\221",
  "\222", "\223", "\224", "\225", "\226", "\227", "\230",
  "\231", "\233", "\235", "\240", "\241", "\242", "\243",
  "\244", "\245", "\246", "\247", "\250", "\251", "\252",
  "\253", "\254", "\255", "\256", "\257", "\260", "\261",
  "\262", "\263", "\264", "\265", "\266", "\267", "\270",
  "\271", "\272", "\273", "\274", "\275", "\276", "\277",
  "\327", "\367")
```

Note

The set of characters allowed or disallowed in object names is not affected by the current locale. The functions `deparse()`, `parse()`, `dump()`, `dput()`, `source()`, `data.dump()`, and `data.restore()` are also unaffected by the choice of locale.

In PostScript Output

The `postscript()` function now uses the Latin1 encoding of its standard fonts by default. This means you can more easily use non-English, Western European characters. If you previously used octal escapes like `"\267"` to get characters in the upper half of the standard PostScript encoding (`"\267"` was the bullet character), you must either change such code (`"\200"` is now the bullet) or use the arguments `setfont=ps.setfont.std` and `bullet=ps.bullet.std` in calls to `postscript()`. The Symbol and Zapf family of fonts are not changed. The Latin1 encoding is not quite the ISO 8859-1 standard, in that the bullet character was added at position `"\200"`.

IMPORTING, EXPORTING, AND DISPLAYING NUMERIC DATA

S-PLUS 6.1 supports importing, exporting, and displaying numeric data written in regional notation. This means you can import, export, and display numbers written using decimal markers and thousands separators other than the period and comma, respectively. For example, you can specify a comma as your decimal marker and a period as your digit-grouping symbol, or you can use a period as the decimal marker and an apostrophe as your digit-grouping symbol.

This feature is supported for the functions `scan()`, `print()`, `read.table()`, and `write.table()`. It is also supported for tick labels in graphics. Note, however, that it is *not* supported for use within S-PLUS expressions, within `xlab` and `ylab` in graphics, or by default within the `importData()` and `exportData()` functions.

Hint

The default values for `xlab` and `ylab` do not use the numeric locale information because the default values are made with `deparse()`. However, if you make up your labels with `as.character()` or `paste()`, the locale information *is* used.

Importing and Exporting Data

The functions `importData()` and `exportData()` now have two additional arguments for use in reading and writing ASCII numbers:

- The `decimal.point` argument controls the single character used to mark the decimal place in ASCII numbers. The default is the period (`.`).
- The `thousands.separator` argument controls the single character used as the digit-grouping symbol in ASCII numbers. The default is the comma (`,`).

You can also specify which locale to use within `importData()` and `exportData()` by setting the argument `use.locale` to `T` (the default is `use.locale=F`.)

Displaying Data

The decimal marker used when displaying or reading numbers changes depending on `LC_NUMERIC`. For example, in many European locales, the number pi is written 3,14159 (approximately). This affects how functions such as `format()`, `print()`, `cat()`, `write.table()`, and `html.table()` display numeric data. It also affects how `as.numeric()` and `as.character()` convert between strings and numbers. Other functions like `cut()` and `factor()` (in the labeling of levels) are also affected.

USER-DEFINED HELP

4

Introduction **54**

Creating, Editing, and Distributing a Help File **56**
- Step 1: Creating the Help File 56
- Step 2: Copying the Help File to a "Clean" Directory 57
- Step 3: Running the `CHAPTER` Utility 57
- Step 4: Editing the Help File 57
- Step 5: Running `Splus6 make install.help` 58
- Step 6: Viewing the Help File 59
- Step 7: Distributing the Help File 60

Common Text Formats **61**

Contents of Help Files **63**
- Descriptions of Fields 63
- Special Help Files 77

INTRODUCTION

When an S-PLUS function is made available system-wide, a help file should always accompany it. Without a help file, use of the function is likely to be limited to those in close contact with the person who wrote it.

Starting with S-PLUS 5.1, the S Language version 4 provides the flexibility of creating and editing your own SGML (Standard Generalized Markup Language) help files to document your user-defined functions in S-PLUS. Using the built-in functions distributed in your S-PLUS installation, you can use and distribute help via JavaHelp™, the help system from Sun Microsystems included with S-PLUS 6.*x*. JavaHelp is used to display the `importData` help file, as shown in Figure 4.1.

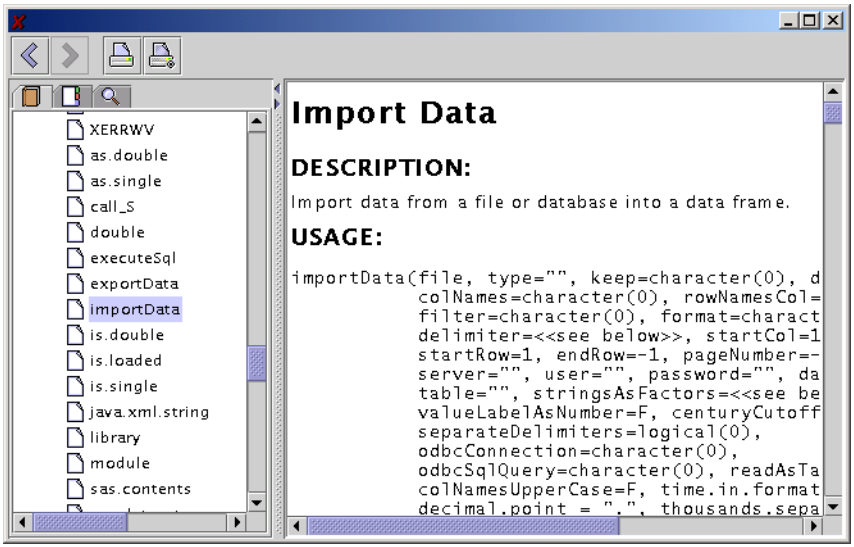

Figure 4.1: *JavaHelp is used to invoke the `importData` help file in S-PLUS 6.1.*

The S-PLUS function `prompt` generates a help file outline for any S-PLUS function or data object you specify. The outline includes the correct syntax for the function, as well as the current definition of the function, and headings under which you can supply the following information:

- What the function does.
- A brief description of the function.
- A description of each argument, with the option of specifying both required and optional arguments.
- A description of the value returned by the function.
- *Side effects*, which are any effects of the function besides the return of a value.
- The *method* behind the function; that is, how the function works.
- Any references to the literature.
- Cross-references to other help files.
- Keywords. Keywords are used by the S-PLUS help system in assembling its **Category** and **Function** menus.

The following sections describe the steps involved in creating, editing, viewing, and distributing your customized help files.

CREATING, EDITING, AND DISTRIBUTING A HELP FILE

Creating a help file for distribution in S-PLUS 6 for UNIX involves the use of S-PLUS commands and commands from a UNIX prompt. In general, the steps are as follows:

1. Use the `prompt` function to generate a SGML file in your working directory.
2. Copy the SGML file to a new directory.
3. Run `Splus6 CHAPTER` on this directory to create an S-PLUS chapter.
4. Edit the SGML file with your editor of choice.
5. Run `Splus6 make install.help` to install the file so it can be accessed by JavaHelp.
6. Launch S-PLUS, attach the S-PLUS chapter, and test the help file by typing

```
> help(myfunc)
```

Repeat steps 4-6 until you are satisfied with the results. If you have a number of files you want to add to your distribution, you can also run this process on a number of help files at once.

Your working directory may change, but as long as you attach the chapter that contains your help files, you can always access them.

To illustrate this process, we create a sample help file, and detail each of the steps involved in distributing it in the following sections.

Step 1: Creating the Help File

Let's suppose you create a new S-PLUS function named `myfunc`:

```
> myfunc
function(x)
return(x * 2)
```

Use the `prompt` function to create a template help file named **myfunc.sgml**:

```
> prompt(myfunc)
created file named myfunc.sgml in the current directory
edit the file according to the directions in the file.
```

By default, this file is placed in your S-PLUS working directory. If you have a number of help files to create, use `prompt` for each function.

Step 2: Copying the Help File to a "Clean" Directory

The next step is to copy the SGML file(s) to a directory that will become an S-PLUS chapter. This chapter becomes the chapter you attach in your S-PLUS session to access your user-defined help files.

From a UNIX prompt, create a new directory and copy the SGML files from the working directory:

```
mkdir myfuncdir
cp myfunc,sgml myfuncdir
```

If you have a number of help files, type

```
cp *,sgml myfuncdir
```

to copy all the SGML files to the directory. Change directories to **myfuncdir** and proceed to the next step.

Step 3: Running the `CHAPTER` Utility

As mentioned in the previous step, the `CHAPTER` utility must be run on the directory so **myfuncdir** can be recognized as an S-PLUS chapter:

```
Splus6 CHAPTER
```

This process creates the chapter's **.Data** directory, which includes the **__Shelp** and **__Hhelp** directories that are required in step 5 to install the files so they can be accessed by JavaHelp.

Step 4: Editing the Help File

You can view and edit the file in your text editor of choice, such as `emacs` or `vi`. For example, if you want to invoke the `vi` editor to edit your file, type the following command:

```
vi myfunc.sgml
```

Your skeleton help file should contain text similar to the following:

```
<!doctype s-function-doc system "s-function-doc.dtd" [
<!entity % S-OLD "INCLUDE">
]
>
<s-function-doc>
<s-topics>
```

```
<s-topic> myfunc </s-topic>
</s-topics>
<s-title>
. . .
```

The first four lines of **myfunc.sgml** are required in all SGML help files for S-PLUS. For this reason, we recommend that you use `prompt` to create a template file rather than write SGML code from scratch.

The `<s-function-doc>` tag begins the contents of the help file; you should see the end tag `</s-function-doc>` at the end of the file. The start and end tags for most fields in an S-PLUS help file are included in the skeleton files that `prompt` creates. If your function does not require a certain field in its help file, you can delete the corresponding tags.

If the text for a given section is particularly long, be sure to start sentences or major phrases on new lines. The SGML parser tends to break when a single line contains too much text. When this occurs, the help file is truncated at the point where the parser breaks and an error message is inserted:

```
Input string too long
```

There is a great deal of information on the meaning and use of the SGML tags in S-PLUS help files, and this is described at length in the section Common Text Formats and the section Contents of Help Files that follow. These sections have been omitted here for brevity.

Step 5: Running `Splus6 make install.help`

Now that the SGML files have been edited, view them in S-PLUS to verify they have the proper content and formatting. This requires another S-PLUS utility to be run, so at a UNIX prompt, type

```
Splus6 make install.help
```

Running the `Splus6 make install.help` utility invokes two other processes:

- `HINSTALL`: This copies **myfunc.sgml** to **.Data/__Shelp/myfunc.sgm** and translates this **.sgm** file to HTML and stores it as **.Data/__Hhelp/myfunc.html**. Note the suffix is **.sgm** and not **.sgml**, which preserves the original file.
- `BUILD_JHELP`: This creates the **__Jhelp** directory and populates the directory with the XML files required to catalog and dispatch JavaHelp. These XML files are described in Table 4.1.

Table 4.1: *Files within the* ***__Jhelp*** *directory.*

File	Description
***.hs**	The help set file, which acts as an identifier to JavaHelp.
***Index.xml**	The index file, which lists all the topics in the help set and is used as the text for the **Index** tab.
***TOC.xml**	The Table-of-Contents file, which lists the topics in the help set by category and is used as the text for the **Contents** tab.
***Map.jhm**	The mapping file, which maps topic names to specific URLs. This is the most important file in the help set; both the Index file and the TOC file rely on it.
mapsHelpSearch	A directory containing the files used by JavaHelp's full text search facility on the **Search** tab.

This process also reveals any errors in SGML and warns you if there are any deficiencies. If the `Splus6 make install.help` should fail in this event, edit your SGML file before proceeding.

Now that the help files can be accessed by JavaHelp, you can view your files in S-PLUS and determine if any changes need to be made.

Step 6: Viewing the Help File

To view the edits in your help file, first launch S-PLUS. Then, attach the chapter and invoke your help file from the **Commands** window:

```
> attach("myfuncdir")
> help(myfunc)
```

You should see the help file for `myfunc` displayed in the help viewer, and you can check for any formatting or content errors. If you need to correct anything, repeat steps 4 through 6 by editing the file, running `Splus6 make install.help`, and viewing the file.

Step 7: Distributing the Help File

You can modify your **S.init** file to attach the chapter where you created your help files each time you start S-PLUS. You can now distribute the files as necessary, which you could package as a compressed archive or self-extracting executable. A utility such as compress or gzip could be used for this purpose.

The remaining sections in this chapter are devoted to the use and format of SGML tags in S-PLUS help files, which are necessary to understand the editing phase of step 4.

COMMON TEXT FORMATS

Table 4.2 lists the most common text formats for S-PLUS SGML files. Note that this list is not exhaustive; in general, there might be many tags to format text in a particular way. We suggest using tags with names that describe the formats themselves. Thus, instead of using <tt> and </tt> for code, we encourage you to use <s-expression> and </s-expression> (or <code> and </code> to save typing).

Table 4.2: *Common text formats in S-PLUS SGML files.*

Format	SGML Tags	Notes
Fixed-width code font	<s-expression>, </s-expression> <code>, </code> <tt>, </tt>	Although <tt> and </tt> still exist in some older help files, please use the other tags instead.
Preformatted text	<pre>, </pre>	This is usually for code samples that extend over multiple lines. With <pre>, you are responsible for line breaks and blank space, as SGML does no formatting for you.
Italics Emphasis	<it>, </it> ,	
Bold	<bf>, </bf>	
Lists Enumerated lists	<descrip>, </descrip> <enum>, </enum> <item>, </item> <tag>, </tag>	The <descrip> and <enum> tags create the lists. Use <item> and </item>, or <tag> and </tag>, to format the individual list elements. See the nlminb help file for an example.

Table 4.2: *Common text formats in S-PLUS SGML files. (Continued)*

Format	SGML Tags	Notes
Line breaks	` `	To include a blank line in a help file, ` ` must be used twice: once for the text line and once for the blank line. The ` ` tag is not needed in preformatted sections.
Paragraph breaks	`<p>`	This is the recommended tag for including blank lines in a help file, as it shows up in `slynx` as well as JavaHelp.

CONTENTS OF HELP FILES

Descriptions of Fields

Each S-PLUS help file is composed of titled sections that appear in the order listed in Table 4.3. We discuss each of these sections below.

Table 4.3: *Titled sections in S-PLUS engine help files. Titles in all capitals appear in a formatted help file. Titles in lowercase letters do not appear in a help file, but the information in those sections do.*

Section Title	Quick Description	SGML Tags
Topic	The name of the function.	<s-topics> </s-topics>
Title	The title that appears at the top of a formatted help file.	<s-title> </s-title>
DESCRIPTION	A short description of the function.	<s-description> </s-description>
USAGE	The function call with all of its arguments.	<s-usage> </s-usage>
REQUIRED ARGUMENTS	Descriptions of arguments that are required by the function.	<s-args-required> </s-args-required>
OPTIONAL ARGUMENTS	Descriptions of arguments that are optional.	<s-args-optional> </s-args-optional>
VALUE	The return value from the function.	<s-value> </s-value>
SIDE EFFECTS	Side effects from the function.	<s-side-effects> </s-side-effects>
GRAPHICAL INTERACTION	A description of graphical interactions expected of the user.	See below.
CLASSES	A description of the classes the function is applicable to, if it is a default method.	See below.

Table 4.3: *Titled sections in S-PLUS engine help files. Titles in all capitals appear in a formatted help file. Titles in lowercase letters do not appear in a help file, but the information in those sections do. (Continued)*

Section Title	Quick Description	SGML Tags
WARNING	Anything the user should be warned about when using the function.	See below.
DETAILS	Descriptions of algorithmic details and implementation issues.	`<s-details>` `</s-details>`
BACKGROUND	Background information on the function or method.	See below.
NOTE	Any information that does not fit into the above categories.	See below.
REFERENCES	Available texts and papers the user can refer to for additional information.	See below.
BUGS	Descriptions of known bugs in the function.	See below.
SEE ALSO	Links to related S-PLUS functions.	`<s-see>` `</s-see>`
EXAMPLES	Coded S-PLUS examples.	`<s-examples>` `</s-examples>`
Keywords	A list of keywords that place the help file in the **Contents** topics of the help system.	`<s-keywords>` `</s-keywords>`

Topic

The topic section contains the function name wrapped in the tags `<s-topic>` and `</s-topic>`. In the **myfunc.sgml** example, the topic looks like:

```
<s-topics>
<s-topic> myfunc </s-topic>
</s-topics>
```

In help files containing multiple functions, each function name should be wrapped in the `<s-topic>` tags. For example, the following is an excerpt from the common SGML file for `cor` and `var`:

```
<s-topics>
   <s-topic>cor</s-topic>
   <s-topic>var</s-topic>
</s-topics>
```

The topic section is not visible in a formatted help file, but is used to index the file in the help system.

Title

The title section contains the title that appears at the top of a formatted help file. For example, the title from the `coxph` SGML file is:

```
<s-title>
Fit Proportional Hazards Regression Model
</s-title>
```

All words in a title should begin in uppercase letters. For clarity, avoid S-PLUS jargon such as function names and class names in the title section. A title should be short enough to fit on one line in the help file.

DESCRIPTION

This section contains a short description of the function. The description in the `coxph` SGML file is:

```
<s-description>
Fits a Cox proportional hazards regression model.
Time dependent variables, time dependent strata, multiple
events per subject, and other extensions are incorporated
using the counting process formulation of Andersen and
Gill.
</s-description>
```

USAGE

This section includes the function call with all of its arguments. You should list optional arguments with the form *`name=default`*. If a default argument is complicated, use the form *`name`*`=<<see below>>` instead and describe the default in the argument's section of the SGML file. Because the angle brackets < and > signify tags in SGML,

however, it is safest to type them as `<` and `>` when tags are not intended. Thus, the form for a complicated default is *name*`=<<see below>>`.

If the help file describes more than one function, the usage for each function should be listed on separate lines. If the argument list for a function is more than one line long, subsequent lines should be indented to one space past the opening parentheses. Use spaces to indent each line instead of tabs. For example, the following is the usage section from the SGML file for `lm`:

```
<s-usage>
<s-old-style-usage>
lm(formula, data=&lt;&lt;see below&gt;&gt;,
   weights=&lt;&lt;see below&gt;&gt;,
   subset=&lt;&lt;see below&gt;&gt;, na.action=na.fail,
   method="qr", model=F, x=F, y=F, contrasts=NULL, ...)
</s-old-style-usage>
</s-usage>
```

The tag `<s-old-style-usage>` preformats the section so that it prints in a fixed-width font. This also causes newlines and blank lines to be recognized in the text. Because of the preformatting, no space is needed around the equals signs for each of the arguments. In addition, please ensure that the width of each line is no more than 60 characters so that the usage section displays nicely in conveniently-sized help windows.

REQUIRED ARGUMENTS

This section lists each required argument wrapped in the tags `<s-arg name="`*name*`">` and `</s-arg>`. The first word in the description of each argument should not be capitalized. For example, the following excerpt lists the three required arguments from the SGML file for `ifelse`:

```
<s-args-required>
<s-arg name="test">
logical object. Missing values <s-object>(NA)</s-object>
are allowed.
</s-arg>
```

```
<s-arg name="yes">
vector containing values to be returned for elements with
<s-expression>test</s-expression> equal to
<s-expression>TRUE</s-expression>.
</s-arg>
<s-arg name="no">
vector containing values to be returned for elements with
<s-expression>test</s-expression> equal to
<s-expression>FALSE</s-expression>.
</s-arg>
</s-args-required>
```

In descriptions of arguments, you should always state whether exceptional values (`NA`, `NaN`, `Inf`, etc.) are treated specially.

OPTIONAL ARGUMENTS

This section lists each optional argument wrapped in the tags `<s-arg name="`*name*`">` and `</s-arg>`. As in the section for required arguments, the first word in each argument's description should not be capitalized, and you should document whether exceptional values are accepted.

VALUE

This section describes the return value of the function. The first word in the description should not be capitalized. Often, the description of a function's return value begins with a phrase similar to:

```
a list containing the following components:
```

In this case, you can use the tags `<s-return-component name="`*name*`">` and `</s-return-component>` to format the individual components of the return list. The first word in the description for each return component should not be capitalized. As an illustration, the value section of the `density` SGML file is provided below.

```
<s-value>
a list with two components, <s-expression>x</s-expression>
and <s-expression>y</s-expression>, suitable for giving as
an argument to one of the plotting functions.
<s-return-component name="x">
a vector of <s-expression>n</s-expression> points at which
the density is estimated.
</s-return-component>
```

```
<s-return-component name="y">
the density estimate at each <s-expression>x</s-expression>
point.
</s-return-component>
</s-value>
```

You should include arcticles in the uncapitalized sentences of the value section. That is, you should write

```
a list with two components
```

instead of simply

```
list with two components
```

SIDE EFFECTS

Side effects of the function (plotting, changing graphics devices, changing session options, etc.) are described in this section. This is also the appropriate place to describe the lack of side effects if a user might expect one. For instance, the fact that many S-PLUS editing functions do not actually change an object can be documented in the side effects sections of their SGML files.

Any function that updates the object `.Random.seed` must include the following message in its side effects section:

> The function *name* causes the creation of the data set `.Random.seed` if it does not already exist. Otherwise, the value of `.Random.seed` is updated.

GRAPHICAL INTERACTION

If the user is expected to interact with a graphical display, the interaction is described in this section. Help files that currently contain this section include `burl.tree`, `hist.tree`, and `snip.tree`. The graphical interaction section in S-PLUS SGML files does not have a specific tag. Instead, the tags `<s-section name="`*name*`">` and `</s-section>` are used. For example, the following excerpt is from the `hist.tree` SGML file.

```
<s-section name="GRAPHICAL INTERACTION">
This function checks that the user is in split-screen mode.
A dendrogram of <s-expression>tree</s-expression> is
expected to be visible on the current active screen, and a
graphics input device (for example, a mouse) is required.
Clicking the selection button on a node results in the
additional screens being filled with the information
described above. This process may be repeated any number of
```

```
times. Warnings result from selecting leaf nodes. Clicking
the exit button will stop the display process and return the
list described above for the last node selected. See
<s-expression>split.screen</s-expression> for specific
details on graphic input and split-screen mode.
</s-section>
```

CLASSES

This section lists the classes the function is applicable to, if it is a default method. Like the graphical interaction section, the classes section in S-PLUS SGML files does not have a specific tag. Instead, the tags `<s-section name="name">` and `</s-section>` are used. For example, the classes section in the `gamma` help file is:

```
<s-section name="CLASSES">
This function will be used as the default method for classes
that do not inherit a specific method for the function or
for the <tt>Math</tt> group of functions. The result will
retain the class and the attributes. If this behavior is
<em>not</em> appropriate, the designer of the class should
provide a method for the function or for the <tt>Math</tt>
group
</s-section>
```

WARNING

Anything the user should be warned about when using the function should be described here. The warning section in S-PLUS SGML files does not have a specific tag, but uses `<s-section name="name">` and `</s-section>` instead. The following is the warning section from the `gamma` help file:

```
<s-section name="WARNING">
<s-expression>gamma(x)</s-expression> increases very
rapidly with <s-expression>x</s-expression>. Use
<s-expression>lgamma</s-expression> to avoid overflow.
</s-section>
```

DETAILS

Algorithmic details and implementation issues are described in this section. For example, the details section of the `density` help file explains the smoothing algorithm implemented in the function:

```
<s-details>
. . .
These are kernel estimates. For each
<s-expression>x</s-expression> value in the output, the
```

```
window is centered on that <s-expression>x</s-expression>
and the heights of the window at each datapoint are summed.
This sum, after a normalization, is the corresponding
<s-expression>y</s-expression> value in the output. Results
are currently computed to single-precision accuracy only.
</s-details>
```

Details that apply to only one argument should be part of the affected argument's description, and not part of the details section.

BACKGROUND

Background information on the function or method is described in this section. The text here should be aimed at those with complete ignorance on the subject. The background section in S-PLUS SGML files does not have a specific tag, but uses `<s-section name="`*name*`">` and `</s-section>` instead. For example, the background section in the `hclust` help file provides general information on clustering algorithms:

```
<s-section name="BACKGROUND">
Cluster analysis divides datapoints into groups of points
that are "close" to each other. The
<s-expression>hclust</s-expression> function continues to
aggregate groups together until there is just one big
group. If it is necessary to choose the number of groups,
this can be decided subsequently. Other methods (see
<s-expression>kmeans</s-expression>) require that the
number of groups be decided from the start.
<p>
By changing the distance metric and the clustering method,
several different cluster trees can be created from a
single dataset. No one method seems to be useful in all
situations. Single linkage
(<s-expression>"connected"</s-expression>) can work poorly
if two distinct groups have a few "stragglers" between
them.
</s-section>
```

NOTE

Anything that does not fit into one of the above categories can be described in this section. The note section in S-PLUS SGML files does not have a specific tag, but uses `<s-section name="name">` and `</s-section>` instead. The following is a note from the `gamma` help file:

```
<s-section name="NOTE">
See <s-expression>family</s-expression> for the family
generating function <s-expression>Gamma</s-expression> used
with the <s-expression>glm</s-expression> and
<s-expression>gam</s-expression> functions. See
<s-expression>GAMMA</s-expression> for the functions
related to the gamma distribution:
<s-expression>dgamma</s-expression> (density),
<s-expression>pgamma</s-expression> (probability),
<s-expression>qgamma</s-expression> (quantile),
<s-expression>rgamma</s-expression> (sample).
</s-section>
```

REFERENCES

References for functions are listed alphabetically and should follow *The Chicago Manual of Style*. The format for a book reference is similar to:

> Venables, W.N. and Ripley, B.D. (1999). *Modern Applied Statistics with S-PLUS* (3rd ed.). New York: Springer-Verlag, Inc.

The format for a journal article is similar to:

> Andersen, P. and Gill, R. (1982). Cox's regression model for counting processes, a large sample study. *Annals of Statistics* **10**: 1100-1120.

The references section in S-PLUS SGML files does not have a specific tag, but uses `<s-section name="name">` and `</s-section>` instead. The following is an excerpt from the references in the SGML help file for `density`, which cites a number of different works:

```
<s-section name="REFERENCES">
Silverman, B.W. (1986).
<it>Density Estimation for Statistics and Data
Analysis.</it> London: Chapman and Hall.
<p>
Wegman, E.J. (1972).
```

```
Nonparametric probability density estimation.
<it>Technometrics</it> <bf>14</bf>: 533-546.
<p>
Venables, W.N. and Ripley, B.D. (1997)
<it>Modern Applied Statistics with S-PLUS</it> (2nd ed.).
New York: Springer-Verlag.
</s-section>
```

The tag for paragraph breaks `<p>` should be used to separate multiple references. Please try to cite works that are widely available to users.

BUGS

In this section, you should document any known bugs a user might encounter while using the function. The bugs section in S-PLUS SGML files does not have a specific tag, but uses `<s-section name="`*name*`">` and `</s-section>` instead. For example, the following is a bug from the SGML help file for `subplot`:

```
<s-section name="BUGS">
If you request it to draw outside the figure region,
<s-expression>subplot</s-expression> gets very confused
and typically puts the subplot across the whole region.
</s-section>
```

SEE ALSO

This section provides links to related functions. In general, any function you reference in the text of a help file should be linked in this section. The name of each function is wrapped in the tags `<s-function name="`*filename*`">` and `</s-function>`; this provides the hyperlinks in the formatted help file. The *filename* is the name of the installed help file, which is the linked function followed by `.sgm`. For example, the links in the SGML file for `subplot` are:

```
<s-see>
<s-function name="symbols.sgm">symbols</s-function>,
<s-function name="locator.sgm">locator</s-function>,
<s-function name="par.sgm">par</s-function>.
</s-see>
```

Functions that rely on the self-doc mechanism for their help files cannot be linked in this section.

Although newlines are not recognized in the see also section of S-PLUS SGML files, spaces are. Thus, be sure to include spaces between each link, even if you type them on separate lines. In the SGML file for `subplot`, two spaces are included at the end of each line, immediately after the `</s-function>` tags.

EXAMPLES

The examples in this section should help the user understand the function better. The goal is to provide the user with clear examples that are easily copied and run, either from the commands window or from a script window. Therefore, do not include the S-PLUS prompt character > in your examples and comment any output you include. So that the examples are self-contained, use built-in data sets or create simple data sets in the code. For clarity, do not abbreviate argument names in your code, and be sure to test your examples before including them in a help file.

The following is an excerpt from the examples section of the `coxph` help file:

```
<s-examples>
<s-example type=text>
# Create the simplest test data set
test1 &lt;- list(time=c(4,3,1,1,2,2,3),
              status=c(1,1,1,0,1,1,0),
              x=c(0,2,1,1,1,0,0),
              sex=c(0,0,0,0,1,1,1))
# Fit a stratified model
coxph(Surv(time, status) ~ x + strata(sex), test1)
. . .
</s-example>
</s-examples>
```

The tag `<s-example type=text>` preformats the examples section so that it prints in a fixed-width font. This also causes newlines and blank lines to be recognized in the text. Thus, you can include spaces between different examples to enhance readability. So that the examples display nicely in conveniently-sized help windows, please ensure that the width of each line is no more than 60 characters.

In your examples, always use the left assignment operator `<-` instead of the underscore _ for assignments. Because the angle bracket < signifies a tag in SGML, it is safest to type the left assignment operator using `<` instead. Thus, the operator is `<-` in SGML.

Keywords

All help files should have keywords listed at the bottom, immediately before the closing tag `</s-function-doc>`. The help system uses keywords to organize the engine functions in the **Contents** tab of the help window; open the **Language Reference** from the S-PLUS menu and click on the **Contents** tab to see this.

Each keyword should be wrapped in the tags `<s-keyword>` and `</s-keyword>`. For example, the keywords section in the SGML file for `coxph` is:

```
<s-keywords>
<s-keyword>models</s-keyword>
<s-keyword>regression</s-keyword>
<s-keyword>survival4</s-keyword>
</s-keywords>
```

This places the `coxph` function in the following **Contents** topics: Statistical Models, Regression, and Survival Analysis. Table 4.4 lists the current keywords and the help topics they map to.

Table 4.4: *Current keywords and the **Contents** topics they map to.*

Keyword	Topic	Keyword	Topic	Keyword	Topic
`aplot`	Add to Existing Plot	`design`	ANOVA Models	`bootstrap`	Bootstrap Methods
`category`	Categorical Data	`character`	Character Data Operations	`cluster`	Clustering
`complex`	Complex Numbers	`dplot`	Computations Related to Plotting	`menudata`	Data Menu Functions
`wdialogs`	Customizable Dialogs (pre-S-PLUS 4.0)	`wmenus`	Customizable Menus (pre-S-PLUS 4.0)	`attribute`	Data Attributes
`data`	Data Directories	`manip`	Data Manipulation	`sysdata`	Data Sets
`classes`	Data Types	`chron`	Dates Objects	`debugging`	Debugging Tools

Table 4.4: *Current keywords and the* ***Contents*** *topics they map to. (Continued)*

Keyword	Topic	Keyword	Topic	Keyword	Topic
`defunct`	Defunct Functions	`deprecated`	Deprecated Functions	`documentation`	Documentation
`dynamic`	Dynamic Graphics	`error`	Error Handling	`device`	Graphical Devices
`hplot`	High-Level Plots	`file`	Input/Output Files	`iplot`	Interacting With Plots
`interface`	Interfaces to Other Languages	`jackknife`	Jackknife Methods	`libchron`	Library of Chronological Functions
`libcluster`	Library of Clustering Methods	`libmaps`	Library of Maps	`algebra`	Linear Algebra
`list`	Lists	`loess`	Loess Objects	`logic`	Logical Operators
`iteration`	Looping and Iteration	`math`	Mathematical Operations	`array`	Matrices and Arrays
`methods`	Methods and Generic Functions	`misc`	Miscellaneous	`missing`	Missing Values
`mixed`	Mixed Effects Models (version 2)	`nlme3`	Mixed Effects Models (version 3)	`multivariate`	Multivariate Techniques
`nonlinear`	Nonlinear Regression	`nonparametric`	Nonparametric Statistics	`optimize`	Optimization
`ode`	Ordinary Differential Equations	`print`	Printing	`distribution`	Probability Distributions and Random Numbers

Table 4.4: *Current keywords and the* ***Contents*** *topics they map to. (Continued)*

Keyword	Topic	Keyword	Topic	Keyword	Topic
programming	Programming	qc	Quality Control	regression	Regression
tree	Classification and Regression Trees	release	Release Notes	resample	Resampling (Bootstrap, Jackknife, and Permutations)
robust	Robust/ Resistant Techniques	environment	S-PLUS Session Environment	smooth	Smoothing Operations
htest	Statistical Inference	menustat	Statistics Menu Functions	models	Statistical Models
survival4	Survival Analysis	ts	Time Series	trellis	Trellis Displays Library
guifun	User Interface Programming	utilities	Utilities	DOX	Design of Experiments, Response Surfaces, and Robust Design
fracfac	Fractional Factorial Experiments	rsm	Response Surfaces	taguchi	Robust Experimental Design
modgarch	GARCH Module for Modeling Time Series Volatility	geostat	Geostatistical Data Analysis	hexbin	Hexagonal Binning
lattice	Lattice Data Analysis	pointpattern	Point Pattern Analysis	spatialreg	Spatial Regression
spatialstats	Spatial Statistics Module	WAVELETS	Wavelet Analysis of Data, Signals, and Images	swt	Discrete Wavelet Transform Analysis

Table 4.4: *Current keywords and the* ***Contents*** *topics they map to. (Continued)*

Keyword	Topic	Keyword	Topic	Keyword	Topic
`transform1d`	1-D Wavelet and Cosine Transforms	`transform2d`	2-D Wavelet and Cosine Transforms	`conv`	Wavelet Convolutions and Filters
`cpt`	Cosine Packet Analysis	`wpt`	Wavelet Packet Analysis	`crystal`	Wavelet Crystals
`molecule`	Wavelet Molecules and Atoms	`wavemake`	Creating Wavelets, Wavelet Packets, and Cosine Packets	`wavelets`	Wavelets Module Functions
`wavedata`	Wavelets Module Signals, Images, and Datasets				

Special Help Files

Some S-PLUS help files do not fit into the general format described above, and instead require special fields. The two most common types of special help files are those for class objects and data sets. In this section, we briefly list the fields in these types of help files. We do not discuss each field in detail, but refer you to specific help files for more information and example SGML code. As with help files for functions, you can use `prompt` to create template SGML files and delete the tags that are not applicable to your objects.

SV4 Class Objects The SGML sections in an SV4 `class.`*`type`* help file are listed in Table 4.5. For more details, see the SGML files for `class.timeSeries`, `class.vector`, and `class.matrix`.

Table 4.5: *Titled sections in S-PLUS help files for SV4 class objects.*

Section Title	Quick Description	SGML Tags
Topic	The name of the object.	`<s-topics>` `</s-topics>`
Title	The title that appears at the top of a formatted help file.	`<s-title>` `</s-title>`
DESCRIPTION	A short description of the object.	`<s-description>` `</s-description>`
CLASS SLOTS	A list of descriptions for the slots in the object. Each slot can be formatted with the list tags `<s-class-slot name=>` and `</s-class-slot>`.	`<s-slots>` `</s-slots>`
EXTENDS	A list of classes the object extends. Each class is formatted with `<s-contains-class name=>` and `</s-contains-class>`.	`<s-contains>` `</s-contains>`
DETAILS	Descriptions of implementation issues.	`<s-details>` `</s-details>`
NOTE	Any information that does not fit into the above categories.	`<s-section name=>` `</s-section>`
REFERENCES	Available texts and papers the user can refer to for additional information.	`<s-section name=>` `</s-section>`
SEE ALSO	Links to related S-PLUS functions. The function that creates the object should be included in the links.	`<s-see>` `</s-see>`
Keywords	A list of keywords that place the help file in the **Contents** topics of the help system.	`<s-keywords>` `</s-keywords>`

Data Sets

The SGML sections in help files for data sets are listed in Table 4.6. For more details, see the SGML files for `solder` and `kyphosis`.

Table 4.6: *Titled sections in S-PLUS help files for data sets.*

Section Title	Quick Description	SGML Tags
Topic	The name of the data object.	`<s-topics>` `</s-topics>`
Title	The title that appears at the top of a formatted help file. This should not be the name of the object itself.	`<s-title>` `</s-title>`
SUMMARY	A brief description of the experiment that produced the data. The name of the object should be included in this section.	`<s-section name=>` `</s-section>`
DATA DESCRIPTION	A short description of each of the variables in the object.	`<s-description>` `</s-description>`
SOURCE	The original references for the data.	`<s-section name=>` `</s-section>`
WARNING	Anything the user should be warned about when using the data.	`<s-section name=>` `</s-section>`
NOTE	Any information that does not fit into the above categories.	`<s-section name=>` `</s-section>`
SEE ALSO	Links to related S-PLUS functions and data sets.	`<s-see>` `</s-see>`
EXAMPLES	Coded S-PLUS examples using the data.	`<s-examples>` `</s-examples>`
Keywords	A list of keywords that place the help file in the **Contents** topics of the help system.	`<s-keywords>` `</s-keywords>`

ORACLE CONNECTIVITY 5

Introduction 82

Reading Data From an Oracle Database 83
The `importData` Function 84
The `openData` Function 87

Writing Data to an Oracle Database 90

Executing Arbitrary SQL Statements 92

INTRODUCTION

S-PLUS 6.1 for Solaris platforms extends connectivity with Oracle database clients. This feature gives you more flexibility in reading data from an Oracle table, inserting data into a table, and running arbitrary SQL queries. This enhanced Oracle connectivity is available from the S-PLUS command line but not from the **Import Data** dialog in the S-PLUS Java GUI.

In this chapter, we briefly discuss the S-PLUS functions involved in Oracle connectivity and provide examples illustrating this feature:

- The section Reading Data From an Oracle Database describes changes made to the `importData` and `openData/readNextDataRows/closeData` set of functions.
- The section Writing Data to an Oracle Database describes changes made to the `exportData` and `openData/writeNextDataRows/closeData` set of functions.
- The section Executing Arbitrary SQL Statements describes `executeSQL`, a new function for executing arbitrary SQL commands.

S-PLUS 6.1 was fully supported and tested with the Oracle 8.1.7 and 9i clients. These clients may be used to communicate with Oracle 8.1.6 and Oracle 9i servers.

Note
The extended Oracle connectivity discussed in this chapter is available on Solaris installations of S-PLUS 6.1 only; Linux installations do not support this feature.

READING DATA FROM AN ORACLE DATABASE

The primary functions for reading data from an Oracle database are:

- `importData`, to import data directly, or
- `readNextDataRows` (in conjunction with `openData` and `closeData`), to read sequential blocks of data from a table in the database.

The `importData` and `openData` functions have been modified in S-PLUS 6.1 to enhance Oracle connectivity on Solaris installations of S-PLUS. In particular, a new `sqlQuery` argument has been added to both functions.

In this section, we describe the changes to `importData` and `openData` and provide examples illustrating them. In many of the examples, we use the following hypothetical table **emp** from an Oracle database named **ORACLEDB**. This table describes employees at an imaginary company and includes fields for employee number, name, job, manager number, hire date, salary, commission, and department number.

	EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
1	7369	SMITH	CLERK	7902	12/17/1980 00:00:00.000	800	NA	20
2	7499	ALLEN	SALESMAN	7698	02/20/1981 00:00:00.000	1600	300	30
3	7521	WARD	SALESMAN	7698	02/22/1981 00:00:00.000	1250	500	30
4	7566	JONES	MANAGER	7839	04/02/1981 00:00:00.000	2975	NA	20
5	7654	MARTIN	SALESMAN	7698	09/28/1981 00:00:00.000	1250	1400	30
6	7698	BLAKE	MANAGER	7839	05/01/1981 00:00:00.000	2850	NA	30
7	7782	CLARK	MANAGER	7839	06/09/1981 00:00:00.000	2450	NA	10
8	7788	SCOTT	ANALYST	7566	04/19/1987 00:00:00.000	3000	NA	20
9	7839	KING	PRESIDENT	NA	11/17/1981 00:00:00.000	5000	NA	10
10	7844	TURNER	SALESMAN	7698	09/08/1981 00:00:00.000	1500	0	30
11	7876	ADAMS	CLERK	7788	05/23/1987 00:00:00.000	1100	NA	20
12	7900	JAMES	CLERK	7698	12/03/1981 00:00:00.000	950	NA	30
13	7902	FORD	ANALYST	7566	12/03/1981 00:00:00.000	3000	NA	20
14	7934	MILLER	CLERK	7782	01/23/1982 00:00:00.000	1300	NA	10

The `importData` Function

The principal tool for importing data from the S-PLUS command line is `importData`. This function accepts a `type` argument, which is a character string specifying the file type of the data to be imported. In older versions of S-PLUS, it is possible to specify `type="oracle"` to import data directly from an Oracle database. When specifying `type="oracle"`, you can define the appropriate Oracle server, user name, password, and table with the `server`, `user`, `password`, and `table` arguments, respectively. The contents of the designated table are imported into S-PLUS and returned as a data frame.

In S-PLUS 6.1, you can also specify the new `sqlQuery` argument when `type="oracle"`, which allows you to execute arbitrary SQL queries with `importData`. This allows more flexibility in the structure of data you can read from Oracle databases. The `sqlQuery` argument must be a single character vector; for example, `"select * from mytable where Type = 'Van'"`. S-PLUS pastes multi-element character vectors together using newlines as separators.

The return value from a call to `importData` that includes `sqlQuery` is the result from the query as an S-PLUS data frame. If the query produces no output, the return value is `NULL`. Any errors or warnings from the database that the query causes are stored as a `"message"` attribute to the return object; messages from the database do not cause errors in the S-PLUS engine.

The `server`, `user`, and `password` arguments to `importData` remain identical in function in S-PLUS 6.1. You can still provide the `table` argument if you do not specify `sqlQuery`; providing both arguments causes S-PLUS to return an error. To illustrate, the following two argument settings are equivalent:

- `table = "mytable"`
- `sqlQuery = "select * from mytable"`

Examples

The following two commands are equivalent and return all data from the **emp** table.

```
# Using the syntax table="emp".
> importData(type="oracle", user="scott", password="tiger",
+ table="emp", server="ORACLEDB")
```

```
# Using the sqlQuery argument.
> importData(type="oracle", user="scott", password="tiger",
+ server="ORACLEDB", sqlQuery = "SELECT * FROM emp")
```

The next command returns the data from **emp** corresponding to employees who work as clerks.

```
> importData(type="oracle", user="scott",
+ password="tiger", server="ORACLEDB",
+ sqlQuery = "SELECT * FROM emp WHERE job = 'CLERK'")
  EMPNO  ENAME  JOB    MGR  HIREDATE                SAL  COMM  DEPTNO
1  7369  SMITH  CLERK  7902  12/17/1980 00:00:00.000  800    NA      20
2  7876  ADAMS  CLERK  7788  05/23/1987 00:00:00.000 1100    NA      20
3  7900  JAMES  CLERK  7698  12/03/1981 00:00:00.000  950    NA      30
4  7934  MILLER CLERK  7782  01/23/1982 00:00:00.000 1300    NA      10
```

The next command selects only the **EMPNO** field from the table.

```
> importData(type="oracle", user="scott", password="tiger",
+ server="ORACLEDB", sqlQuery = "SELECT empno FROM emp")
   EMPNO
1   7369
2   7499
3   7521
4   7566
5   7654
6   7698
7   7782
8   7788
9   7839
10  7844
11  7876
12  7900
13  7902
14  7934
```

To select all data from **emp** and order the rows alphabetically according to employee name, type the following:

```
> importData(type="oracle", user="scott",
+ password="tiger", server="ORACLEDB",
+ sqlQuery = "SELECT * FROM emp ORDER BY ename")
```

```
   EMPNO  ENAME  JOB        MGR  HIREDATE                 SAL  COMM DEPTNO
1   7876  ADAMS  CLERK     7788  05/23/1987 00:00:00.000  1100    NA     20
2   7499  ALLEN  SALESMAN  7698  02/20/1981 00:00:00.000  1600   300     30
3   7698  BLAKE  MANAGER   7839  05/01/1981 00:00:00.000  2850    NA     30
4   7782  CLARK  MANAGER   7839  06/09/1981 00:00:00.000  2450    NA     10
5   7902  FORD   ANALYST   7566  12/03/1981 00:00:00.000  3000    NA     20
6   7900  JAMES  CLERK     7698  12/03/1981 00:00:00.000   950    NA     30
7   7566  JONES  MANAGER   7839  04/02/1981 00:00:00.000  2975    NA     20
8   7839  KING   PRESIDENT   NA  11/17/1981 00:00:00.000  5000    NA     10
9   7654  MARTIN SALESMAN  7698  09/28/1981 00:00:00.000  1250  1400     30
10  7934  MILLER CLERK     7782  01/23/1982 00:00:00.000  1300    NA     10
11  7788  SCOTT  ANALYST   7566  04/19/1987 00:00:00.000  3000    NA     20
12  7369  SMITH  CLERK     7902  12/17/1980 00:00:00.000   800    NA     20
13  7844  TURNER SALESMAN  7698  09/08/1981 00:00:00.000  1500     0     30
14  7521  WARD   SALESMAN  7698  02/22/1981 00:00:00.000  1250   500     30
```

To select the subset of data corresponding to employees who make more than the average salary, type the command below. In this command, only the fields **DEPTNO**, **ENAME**, and **SAL** are returned and the results are ordered by department number.

```
> importData(type="oracle", user="scott", password="tiger",
+ server="ORACLEDB", sqlQuery =
+ "SELECT deptno, ename, sal FROM emp x WHERE sal >
+ (SELECT AVG(sal) FROM emp WHERE x.deptno = deptno)
+ ORDER BY deptno")
   DEPTNO  ENAME   SAL
1      10  KING   5000
2      20  JONES  2975
3      20  SCOTT  3000
4      20  FORD   3000
5      30  ALLEN  1600
6      30  BLAKE  2850
```

The final command below returns the minimum and maximum salaries from each department as well as a count of the number of employees in each department.

```
> importData(type="oracle", user="scott", password="tiger",
+ server="ORACLEDB", sqlQuery =
+ "SELECT deptno, MIN(sal), MAX (sal),
+ COUNT(*) FROM emp GROUP BY deptno")
```

```
   DEPTNO  MIN.SAL.  MAX.SAL.  COUNT...
1      10     1300      5000         3
2      20      800      3000         5
3      30      950      2850         6
```

To capture the results from any of these commands in S-PLUS objects, simply assign the output from the commands. For example:

```
> salary.extremes <- importData(
+ type="oracle", user="scott", password="tiger",
+ server="ORACLEDB", sqlQuery =
+ "SELECT deptno, MIN(sal), MAX (sal),
+ COUNT(*) FROM emp GROUP BY deptno")
```

The `openData` Function

The `openData` function opens an external file so the data in the file may be read into S-PLUS. It returns a *data handle*, which is an S-PLUS object that can be passed to `readNextDataRows` to import sequential blocks of data from the file. Together, `openData`, `readNextDataRows`, and `closeData` (the function for closing the external file when all data is imported) provide the capability for reading arbitrarily large data sets into S-PLUS.

The arguments to `openData` are structured similarly to those for `importData`. In particular, `openData` accepts a `type` argument you can set to `"oracle"` for importing data from an Oracle database. In S-PLUS 6.1, `openData` also accepts the new argument `sqlQuery`, which allows you to execute arbitrary SQL statements when importing Oracle data. As with `importData`, the `sqlQuery` argument to `openData` must be a single character vector and cannot be specified in conjunction with the `table` argument.

Note

When `openData` is called with the `table` argument, a new table is created if one by the specified name does not already exist.

Examples

First, we create a table named **test_1** in the Oracle database **ORACLEDB** that is a copy of the built-in data set `fuel.frame`. See the section Writing Data to an Oracle Database for information on the `exportData` function.

```
> exportData(fuel.frame, type="oracle", user="scott",
+ password="tiger", table="test_1", server="ORACLEDB")
[1] 60
```

Use `openData` to create a data handle that reads 10 rows at a time from the **test_1** table:

```
> dh <- openData(type="oracle", rowsToRead=10,
+ openType="read", user="scott", password="tiger",
+ table="test_1", server="ORACLEDB")
```

Read the first 10 rows from the table:

```
> readNextDataRows(dh)
   WEIGHT  DISP.  MILEAGE      FUEL  XTYPE
1    2560    97        33  3.030303  Small
2    2345   114        33  3.030303  Small
3    1845    81        37  2.702703  Small
4    2260    91        32  3.125000  Small
5    2440   113        32  3.125000  Small
6    2285    97        26  3.846154  Small
7    2275    97        33  3.030303  Small
8    2350    98        28  3.571429  Small
9    2295   109        25  4.000000  Small
10   1900    73        34  2.941176  Small
```

Read the second 10 rows from the table:

```
> readNextDataRows(dh)
   WEIGHT  DISP.  MILEAGE      FUEL  XTYPE
1    2390    97        29  3.448276  Small
2    2075    89        35  2.857143  Small
3    2330   109        26  3.846154  Small
4    3320   305        20  5.000000  Sporty
5    2885   153        27  3.703704  Sporty
6    3310   302        19  5.263158  Sporty
```

```
7    2695   133        30  3.333333  Sporty
8    2170    97        33  3.030303  Sporty
9    2710   125        27  3.703704  Sporty
10   2775   146        24  4.166667  Sporty
```

As with `importData`, assign the output from the commands to capture the results in S-PLUS objects. For example:

```
> rows.21.thru.30 <- readNextDataRows(dh)
```

When all data has been read into S-PLUS, close the connection to the Oracle database:

```
> closeData(dh)
[1] T
```

WRITING DATA TO AN ORACLE DATABASE

The primary functions for writing data to an Oracle database are:

- `exportData`, to export data directly, or
- `writeNextDataRows` (in conjunction with `openData` and `closeData`), to write sequential blocks of data to a table in the database.

As its name suggests, the `exportData` function is the counterpart to `importData`. In S-PLUS 6.1, the arguments in Table 5.1 have been added to `exportData` to enhance Oracle connectivity on Solaris installations of S-PLUS. These new arguments are analogs to the corresponding `importData` arguments and function in generally the same way to write data to an Oracle database.

Likewise, the `writeNextDataRows` function is the counterpart to `readNextDataRows`. You can use `writeNextDataRows` to insert data into a table in an Oracle database. Before calling `writeNextDataRows` to do this, however, you must open a connection to the table using `openData` with the `table` argument, not the `sqlQuery` argument (see page 87 for details).

Table 5.1: *New arguments to `exportData` that support Oracle connectivity.*

Argument Name	Description
`server`	A character string specifying the database server.
`user`	A character string specifying the user name.
`password`	A character string specifying the user's password for accessing the database.
`database`	A character string specifying the database. This must be the empty string `""` when `type="oracle"`.
`table`	A character string specifying the name of the table to write to when `type="oracle"`. If a table by the specified name already exists, `exportData` appends data to it. If the table does not exist, `exportData` creates it.

When writing to a table, `exportData` creates a new table if one by the specified name does not exist; otherwise, it appends rows of data to the existing table. If some or all of the column names in the exported S-PLUS data frame do not correspond to fields in the table, `exportData` adds additional fields as necessary. The same is true for `openData` and `writeNextDataRows`: when `openData` is called with the `table` argument, it creates a new table if one by the specified name does not exist. The `writeNextDataRows` appends rows of data to the table, adding new fields as necessary. You cannot use `exportData` or `writeNextDataRows` to overwrite an existing Oracle table.

See the examples in the section Executing Arbitrary SQL Statements on page 92 for examples using both `exportData` and `writeNextDataRows`.

Hints

Another way of adding rows to an existing table is to write to a temporary table and then use an `INSERT` statement with `executeSQL` to merge the temporary table to an existing one. This may reduce issues with a database being locked due to another process writing to it.

To overwrite an existing table in an Oracle database, use the `executeSQL` function to drop the table first and then use `exportData` (or `openData` with `writeNextDataRows`) to create a new one.

See the section Executing Arbitrary SQL Statements below for information on `executeSQL`.

EXECUTING ARBITRARY SQL STATEMENTS

S-PLUS 6.1 includes a new function named `executeSQL` for executing arbitrary SQL statements. This function has the arguments listed in Table 5.2. You can use `executeSQL` as an interface to your Oracle database for performing functions such as creating blank tables and inserting rows into an existing table.

Table 5.2: *Arguments for the `executeSQL` function.*

Argument Name	Description
`type`	A character string, either `"oracle"` or `"odbc"`, specifying the type of database connection. Oracle connectivity is supported only in Solaris versions of S-PLUS 6.1 and ODBC is supported only in Windows versions.
`server`	A character string specifying the database server.
`user`	A character string specifying the user name.
`password`	A character string specifying the user's password for accessing the database.
`database`	A character string specifying the database. This must be the empty string `""` when `type="oracle"`.
`odbcConnection`	A character string containing an ODBC connection string when `type="odbc"`. This argument is functional only in Windows versions of S-PLUS 6.1.
`sqlQuery`	A character string specifying the SQL query to execute.

The return value from `executeSQL` is a character vector containing any messages from the database, or `NULL` if no messages were generated. Errors from the Oracle database do not cause errors in the S-PLUS engine.

The `executeSQL` function discards all data returned by the query. You can use `importData` or `openData` to import data from an Oracle database; see the section Reading Data From an Oracle Database on page 83 for details. Whenever data is discarded, `executeSQL` prints a warning message as notification.

Examples

In the following sequence of commands, we first create a table named **test_2** in the Oracle database **ORACLEDB** and then write the first 20 rows of `fuel.frame` to it. To do this, we use the `openData` function to open a connection to the database, `writeNextDataRows` to write the data to the table, and `closeData` to close the connection.

```
> executeSQL(type="oracle", user="scott",
+ password="tiger", server="ORACLEDB",
+ sqlQuery = "create table test_2(x number)")
> dh <- openData(type="oracle", openType="write",
+ user="scott", password="tiger",
+ table="test_2", server="ORACLEDB")
> writeNextDataRows(fuel.frame[1:10,], dh)
10
> writeNextDataRows(fuel.frame[11:20,], dh)
10
> closeData(dh)
[1] T
```

The next command uses `exportData` to write the next 30 rows of `fuel.frame` to **test_2**. This command does not overwrite the table, but instead appends the rows to it.

```
> exportData(fuel.frame[21:50,], type="oracle",
+ user="scott", password="tiger", table="test_2",
+ server="ORACLEDB")
[1] 40
```

Finally, we use an `INSERT` statement to append the 50 rows in **test_2** to the **test_1** table; see the examples on page 88 for instructions on creating **test_1**. This illustrates a different way of adding rows to an existing table in an Oracle database, which can reduce issues with the database being locked due to another process writing to it.

```
> executeSQL(type="oracle", user="scott",
+ password="tiger", server="ORACLEDB",
+ sqlQuery = "INSERT INTO test_1 SELECT * FROM test_2")
NULL
```

INDEX

Symbols

.Call interface 8, 21, 23, 25, 27, 29, 34, 40

A

arithmetic operators 36
Assign 22
assignment operators 31, 32, 33

B

BUILD_JHELP 58

C

C++
    See CONNECT/C++
catch block 29
characters
    allowed in variable names 48
    not allowed in variable names 49
character sets 8, 44, 46, 48
    ISO 8859-1 8, 45, 48, 49
classes
    CONNECT/C++ 20, 21, 25, 28, 31, 32, 35, 36, 38, 40, 41
    connection 29
    data object 28
    evaluator 29, 30
    function evaluation 28
class library 20, 28, 31
C locale 45, 46
closeData function 82, 83, 87, 90, 93
collating 44
collation sequences 8, 45, 46
Commit 39
CONNECT/C++ 20
    .Call interface 21, 23, 25, 27, 29, 34, 40
    Assign 22
    catch block 29
    classes 20, 21, 25, 28, 31, 32, 35, 36, 38, 40, 41
        connection 29
        data object 28
        evaluator 29, 30
        function evaluation 28
    class library 20, 28, 31
    Commit 39
    constructors 28, 31, 32, 39, 40
    converting objects 34
    Create 22, 28, 38, 39, 40
    CSParray 35
    CSPengineConnect 21, 29, 39, 40
    CSPevaluator 29, 30, 40
    CSPnumeric 21, 34, 35, 38, 39, 41
    CSPobject 28, 31, 32, 33, 34, 36, 38, 39, 40, 41
    CSPproxy 36
    CSPvector 35
    destructors 31
    generating functions 31

IsValid 31, 38
make utility 26, 27
methods 25, 28, 29, 31, 39
objects
named 38, 39, 40
unnamed 40
OnModify 38, 39
OnRemove 38, 39, 40
operators
arithmetic 36
assignment 31, 32, 33
conversion 34
overloading 33, 35, 36
subscripting 35, 36
printing 38
reference counts 31, 32, 33, 34
Remove 38, 39
S.so 27
S_EVALUATOR 25
s_object 25, 28, 34, 38, 39, 40
sconnect.h 21, 25
-sconnectlib 27
See C++
SyncParseEval 22
try block 25, 29
try-catch block 25
connection classes 29
connectivity, Oracle 82
See Oracle connectivity
constructors 28, 31, 32, 39, 40
context-sensitive help 10
conventions, typographic 18
conversion operators 34
converting objects 34
Create 22, 28, 38, 39, 40
CSParray 35
CSPengineConnect 21, 29, 39, 40
CSPevaluator 29, 30, 40
CSPnumeric 21, 34, 35, 38, 39, 41
CSPobject 28, 31, 32, 33, 34, 36, 38, 39, 40, 41
CSPproxy 36
CSPvector 35

D

data object classes 28
decimal.point argument 50
decimal markers 8, 45, 50, 51
destructors 31
digit-grouping symbols 50
displaying numerics 8, 44, 45, 50, 51

E

evaluator classes 29, 30
executeSQL function 82, 91, 92, 93
exportData function 50, 82, 88, 90, 91, 93
exporting numerics 8, 44, 50

F

function evaluation classes 28

G

Gauss-Seidel method 23, 24, 25
generating functions 31
globalization 8

H

help 10, 54
? function 13
at the command line 13
context-sensitive 10
from the graphical user interface 10
help.off function 10
help.start function 10
help function 13
Help menu 10
help window
navigation pane 10, 12
Index page of 12
Search page of 12
Table of Contents page of 12

toolbar 10, 11
buttons on 11
topic pane 10, 12
keywords 12
Language Reference 74
help.off function 10
help.start function 10
help files, user-defined 9, 54, 57
BUILD_JHELP 58
HINSTALL 58
keywords 55, 64, 74, 78, 79
prompt function 55, 56, 57, 58, 77
Splus6 make install.help 56, 58, 59, 60
text formats for 61
titled sections in 63, 78, 79
HINSTALL 58

I

importData function 50, 82, 83, 84, 87, 89, 90, 93
importing numerics 8, 44, 50
install.help, Splus6 make 56, 58, 59, 60
installation 4
ISO 8859-1 character set 8, 45, 48, 49
IsValid 31, 38

J

Java
runtime environment 4
JavaHelp
See help
JRE 4

K

keywords 12, 55, 64, 74, 78, 79

L

Language Reference 74
Latin1 44, 48, 49
locales 8, 9, 44, 45, 48, 49
C 45, 46
changing 46
setting 46
Sys.getlocale function 46
Sys.setlocale function 46
Sys.withlocale function 46

M

make 26, 27
makefile 26
make install.help, Splus6 56, 58, 59, 60
manuals, online 13
viewing 13
markers, decimal 8, 45, 50, 51
methods, CONNECT/C++ 25, 28, 29, 31, 39

N

named objects 38, 39, 40
names, variables
characters allowed in 48
characters not allowed in 49
navigation pane, help window 10, 12
Index page of 12
Search page of 12
Table of Contents page of 12
numerics
displaying 8, 44, 45, 50, 51
exporting 8, 44, 50
importing 8, 44, 50

O

objects
named 38, 39, 40
unnamed 40
OnModify 38, 39
OnRemove 38, 39, 40

openData function 82, 83, 87, 88, 90, 91, 93
operators
  arithmetic 36
  assignment 31, 32, 33
  conversion 34
  overloading 33, 35, 36
  subscripting 35, 36
Oracle 9
Oracle connectivity 82, 83, 90, 92
  closeData function 82, 83, 87, 90, 93
  executeSQL function 82, 91, 92, 93
  exportData function 82, 88, 90, 91, 93
  importData function 82, 83, 84, 87, 89, 90, 93
  openData function 82, 83, 87, 88, 90, 91, 93
  reading data 83
  readNextDataRows function 82, 83, 87, 90
  SQL queries 82, 84, 87, 92
  writeNextDataRows function 82, 90, 91, 93
  writing data 90
order, sort 45
order function 46, 47
overloading operators 33, 35, 36

P

platforms, supported 3
postscript function 49
prompt function 55, 56, 57, 58, 77

Q

queries, SQL 9

R

readNextDataRows function 82, 83, 87, 90
reference counts 31, 32, 33, 34
Regional Options, Windows 8
regional settings 46
Regional Settings, Windows 8
Remove 38, 39
resources 10
runtime environment, Java 4

S

S.init file 46, 60
S.so 27
S_EVALUATOR 25
s_object 25, 28, 34, 38, 39, 40
sconnect.h 21, 25
sequences, collation 8, 45, 46
settings, regional 46
Setup, S-PLUS 44
sort function 46, 47
sorting 44
sort order 45
S-PLUS
  books using
    graphical techniques 17
    introductory 15
    S-PLUS programming 17
    statistics and data analysis
      general 15
      specialized 16
  installing 4
  running 5
  Setup 44
  starting 5
    with command line supporting Java 6
    with command line without Java 5
    with graphical user interface 6
  training courses in 14
  Web site 14
SQL queries 9, 82, 84, 87, 92
Standard Generalized Markup Language (SGML) 54
subscripting operators 35, 36

support, technical 14, 15
swap space 3
symbols, digit-grouping 50
SyncParseEval 22
Sys.getlocale function 46
Sys.setlocale function 46
Sys.withlocale function 46
system requirements 3

T

technical support 14, 15
thousands.separator argument 50
thousands separators 8
    See digit-grouping symbols
toolbar, help window 10, 11
    buttons on 11
topic pane, help window 10, 12
training courses, S-PLUS 14
try block 25, 29
try-catch block 25
typographic conventions 18

U

unnamed objects 40
user-defined help files
    See help files, user-defined

V

variable names
    characters allowed in 48
    characters not allowed in 49

W

Windows Regional Options 8
Windows Regional Settings 8
writeNextDataRows function 82, 90, 91, 93